Michel Rueher (Ed.)

Principles and Practice
of Constraint Programming

22nd International Conference, CP 2016
Toulouse, France, September 5–9, 2016
Proceedings

Springer

Lecture Notes in Computer Science $\quad$ 9892

More information about this series at http://www.springer.com/series/7408

Michel Rueher (Ed.)

Principles and Practice
of Constraint Programming

22nd International Conference, CP 2016
Toulouse, France, September 5–9, 2016
Proceedings

Springer

Editor
Michel Rueher
University of Nice Sophia Antipolis
Sophia Antipolis
France

Printed on acid-free paper

Preface

This volume contains the proceedings of the 22nd International Conference on the Principles and Practice of Constraint Programming (CP 2016), which was held in Toulouse, France, during September 5–9, 2016. Detailed information about the conference is available at http://cp2016.a4cp.org.

The CP conference is the annual international conference on constraint programming. It is concerned with all aspects of computing with constraints, including theory, algorithms, environments, languages, models, systems, and applications such as decision making, resource allocation, scheduling, configuration, and planning. The CP community is very keen to ensure it remains open to interdisciplinary research at the intersection between constraint programming and related fields. Hence, in addition to the usual technical and application tracks, the CP 2016 conference featured the following new thematic tracks: "Computational Sustainability", "CP and Biology", "Preferences, Social Choice and Optimization", and "Testing and Verification". Each track had a specific sub-committee to ensure that specialist reviewers from the relevant domains vetted papers in the respective tracks. CP 2016 also introduced a challenge based on a realistic industrial-grade optimization problem.

For the purpose of the conference's scientific programming, we invited submissions to all tracks, and we received 154 submissions, including 17 submissions for the "Journal-First and Sister Conferences Paper" track. The review process for CP 2016 relied on a multi-tier approach involving one senior Program Committee, dedicated regular Program Committees for all tracks, along with a set of additional reviewers recruited by Program Committee members. Authors submitted either long or short papers. All submissions were assigned to a senior Program Committee member and three members of the relevant track Program Committee. Authors were given an opportunity to respond to reviews before a detailed discussion was undertaken at the level of the Program Committees, overseen by the program chair, the senior Program Committee member, and the track chairs. The "Journal-First and Sister Conferences" track gives an opportunity to discuss important results in the area of constraint programming that appeared recently in relevant journals and sister conferences. Submissions were evaluated by a separate Program Committee for relevance and significance. A meeting of the senior Program Committee was held at Banff –with participation by video conference– at the end of May, chaired by the program chair, where the reviews, author feedback, and discussions on every paper were revisited in detail. The result of this was that the acceptance rate for the technical track was a little under 45%. The senior Program Committee awarded the Best Conference Paper Prize to Krishnamurthy Dvijotham, Pascal Van Hentenryck, Michael Chertkov, Sidhant Misra, and Marc Vuffray for "Graphical Models for Optimal Power Flow", the Distinguished Conference Paper Prize to David Manlove, Iain McBride, and James Trimble for "Almost-Stable Matchings in the Hospitals/Residents Problem with Couples", the Best Student Paper Prize to Clément Carbonnel for "The Dichotomy for Conservative Constraint

Satisfaction Is Polynomially Decidable", and the Distinguished Student Paper Prize to Kyle E.C. Booth, Goldie Nejat, and J. Christopher Beck for "A Constraint Programming Approach to Multi-Robot Task Allocation and Scheduling in Retirement Homes". The program chair, the journal publication fast track chair, Willem-Jan van Hoeve, and the *Constraints* journal editor-in-chief, Michela Milano, also invited four papers from the technical and application tracks for direct publication in that journal. These were presented at the conference like any other paper and they appear in the proceedings as a one-page abstract.

The conference program featured four invited talks by Pascal Van Hentenryck, David Manlove, Andrey Rybalchenko, and Zico Kolter. This volume includes one-page abstracts of their talks. The conference also featured four tutorials and six satellite workshops, whose topics are listed in this volume. The conference also featured four tutorials and five satellite workshops, whose topics are listed in this volume. The Doctoral Program gave PhD students an opportunity to present their work to more senior researchers, to meet with an assigned mentor for advice on their research and early career, to attend special tutorials, and to interact with one another.

I am grateful to many people who made this conference such a success. First of all, to the authors who provided excellent material to select from. Then to the members of the Program Committees and additional reviewers who worked hard to provide constructive, high-quality reviews, to members of the senior Program Committee who helped me ensure that each paper was adequately discussed, wrote meta-reviews for their assigned papers, and participated in live remote deliberations — for some, quite early or late in the day. Of course there is a whole team standing with me, who chaired various aspects of the conference: Thomas Schiex (Conference Chair), Laurent Michel (Application Track Chair), Carla Gomes, Michela Milano and Christine Solnon (Computational Sustainability Track Chairs), Agostino Dovier and Alessandro Dal Palù (CP and Biology Track Chairs), Charlotte Truchet (Music Track Chair), Andreas Podelski and Arnaud Gotlieb (Testing and Verification Track Track Chair), Michela Milano (Published Journal and Sister Conferences Paper Track Chair), Willem-Jan van Hoeve (Journal Publication Fast-Track Chair), Pierre Flener (Workshop and Tutorial Chair), Pierre Schaus (ACP Challenge Chair), Tias Guns and Laura Climen (Doctoral Program Chairs), Helmut Simonis (Industry Outreach Chair), and Louis Martin Rousseau (Publicity Chair).

The conference would not have been possible without the great job done by all the people involved in the local organization: Lotte Berghman, Paul Gaborit, Simon de Givry, Emmanuel Hebrard, Élise Vareilles, Matthias Zytnicki, Alain Pérault, Fabienne Ayrignac, Marie-José Huguet, David Allouche, Nathalie Julliand, George Katsirelos, Pierre Lopez, Alain Haït, Frédéric Maris, Cédric Pralet, Gérard Verfaillie, Nicolas Barnier, Frédéric Messine, and Vincent Vidal. We also want to thank the institutions that supported them during the organization: the Toulouse Business School (for hosting us too), the "École nationale supérieure des mines d'Albi-Carmaux" (also for website hosting and management), the National Institute for Agronomical Research (INRA – MIAT), the National Center for Scientific Research (CNRS – LAAS), the National Applied Sciences Institute (INSA Toulouse), the Higher Institute for Aeronautics and Space (ISAE), the Toulouse Computing Research Institute (IRIT – Toulouse University), the National Office for Aerospace Research and Studies (ONERA), the Civil

July 2016   Michel Rueher

Tutorials and Workshops

Tutorials

Topics in Computational Sustainability

Carla Gomes   Cornell University, USA

Constraint Programming in Music

Charlotte Truchet   University of Nantes, France

Social Choice

Francesca Rossi   University of Padova, Italy

Automated Program Analysis and Verification

Andreas Podelski   University of Freiburg, Germany

Workshops

Configuration

Élise Vareilles   Mines Albi, France
Lars Hvam   Technical University of Denmark, Denmark
Cipriano Forza   University of Padova, Italy
Caroline Becke   PROS Toulouse, France

Constraint Programming and Artificial Intelligence

Eugene C. Freuder   University College Cork, Ireland

CP Meets Verification

Sébastien Bardin   François Bobot and Nikolai Kosmatov (CEA LIST, France)

Constraint Modelling and Reformulation (ModRef 2016)

Steve Prestwich   University College Cork, Ireland

Constraint-Based Methods for Bioinformatics (WCB'16)

Alessandro Dal Palù   University of Parma, Italy
Agostino Dovier   University of Udine, Italy
Simon de Givry   INRA – MIAT, France

Conference Organization

Program Chair

Michel Rueher   University of Nice Sophia Antipolis, France

Conference Chair

Thomas Schiex   INRA Toulouse, France

Application Track Chair

Laurent Michel   University of Connecticut, USA

Computational Sustainability Track Chairs

Carla Gomes   Cornell University, Ithaca, USA
Michela Milano   University of Bologna, Italy
Christine Solnon   INSA de Lyon, France

CP and Biology Track Chairs

Agostino Dovier   University of Udine, Italy
Alessandro Dal Palù   University of Parma, Italy

Music Track Chair

Charlotte Truchet   University of Nantes, France

Preferences, Social Choice, and Optimization Track Chairs

Francesca Rossi   University of Padova, Italy
Toby Walsh   NICTA, Australia

Testing and Verification Track Chairs

Andreas Podelski   University of Freiburg, Germany
Arnaud Gotlieb   Simula Research Laboratory, Norway

Published Journal Track Chair

Michela Milano   University of Bologna, Italy

Journal Publication Fast-Track Chair

Willem-Jan van Hoeve   Tepper School of Business, Carnegie Mellon University,
                                USA

ACP Challenge Chair

Pierre Schaus   UC Louvain, Belgium

Doctoral Program Chairs

Tias Guns   KU Leuven, Belgium
Laura Climent   Insight Centre for Data Analytics, Ireland

Workshop and Tutorial Chair

Pierre Flener   Uppsala University, Sweden

Industry Outreach Chair

Helmut Simonis   Insight Centre for Data Analytics, UCC, Ireland

Publicity Chair

Louis-Martin Rousseau   École Polytechnique de Montréal, Canada

Senior Program Committee

Agostino Dovier   University of Udine, Italy
Pascal van Hentenryck   University of Michigan, USA
Willem-Jan van Hoeve   Tepper School of Business, Carnegie Mellon University,
                                USA
Carla Gomes   Cornell University, Ithaca, USA
Laurent Michel   University of Connecticut, USA
Michela Milano   University of Bologna, Italy
Nina Narodytska   Samsung Research America, USA
Andreas Podelski   University of Freiburg, Germany
Francesca Rossi   University of Padova, Italy
Thomas Schiex   INRA Toulouse, France
Christine Solnon   INSA de Lyon, France
Charlotte Truchet   University of Nantes, France
Toby Walsh   NICTA, Australia

Technical Track Program Committee

Carlos Ansótegui	Universitat de Lleida, Spain
Fahiem Bacchus	University of Toronto, Canada
Roman Bartak	Charles University in Prague, Czech Republic
Chris Beck	University of Toronto, Canada
Nicolas Beldiceanu	École des Mines de Nantes, France
David Bergman	University of Connecticut, USA
Christian Bessiere	CNRS – University of Montpellier, France
Mats Carlsson	SICS, Sweden
David Cohen	Royal Holloway, University of London, UK
Yves Deville	UC Louvain, Belgium
Bistra Dilkina	Georgia Institute of Technology, USA
Carmen Gervet	University of Montpellier, France
Arnaud Gotlieb	Simula Research Laboratory, Norway
Youssef Hamadi	Algorithmic Trading, UK
Emmanuel Hebrard	LAAS-CNRS, France
John Hooker	Carnegie Mellon University, USA
Hiroshi Hosobe	Hosei University, Japan
Peter Jeavons	University of Oxford, UK
George Katsirelos	INRA Toulouse, France
Yahia Lebbah	University of Oran, Algeria
Christophe Lecoutre	University of Artois, France
Jimmy Lee	The Chinese University of Hong Kong, SAR China
Michele Lombardi	University of Bologna, Italy
Inês Lynce	Tecnico University of Lisbon, Portugal
Amnon Meisels	Ben-Gurion University of the Negev, Israel
Pedro Meseguer	IIIA, University of Barcelona, Spain
Claude Michel	University of Nice Sophia Antipolis, France
Ian Miguel	The University of St. Andrews, UK
Barry O'Sullivan	4C University College Cork, Ireland
Justin Pearson	Uppsala University, Sweden
Laurent Perron	Google, France
Gilles Pesant	École Polytechnique de Montréal, Canada
Justyna Petke	University College London, UK
Andrea Rendl	University of Klagenfurt, Austria
Pierre Schaus	UCLouvain, Belgium
Christian Schulte	KTH Royal Institute of Technology, Sweden
Peter Stuckey	University of Melbourne, Australia
Michael Trick	Carnegie Mellon University, USA
Roland Yap	National University of Singapore, Singapore
Stanislav Zivny	University of Oxford, UK

Application Track Program Committee

Laurent Michel	University of Connecticut, USA
Helmut Simonis	4C, Ireland
Louis-Martin Rousseau	École Polytechnique de Montréal, Canada
Pierre Schaus	UC Louvain, Belgium
Michele Lombardi	University of Bologna, Italy
Philip Kilby	NICTA and the Australian National University, Australia
Pascal Van Hentenryck	University of Michigan, USA
Hadrien Cambazard	G-SCOP, Grenoble INP, CNRS, University Joseph Fourier, France
Andrea Lodi	Polytechnique Montréal, Canada
Chris Beck	University of Toronto, Canada
Barry O'Sullivan	4C, University College Cork, Ireland, Ireland
Claude Michel	I3S - (University of Nice/CNRS), France
Michela Milano	DEIS University of Bologna, Italy
Paul Shaw	IBM, France
Simon De Givry	INRA – UBIA, France

Computational Sustainability Track Program Committee

Romain Billot	LICIT, IFSTTAR, France
Bistra Dilkina	Georgia Institute of Technology, USA
Agostino Dovier	University of Udine, Italy
Serge Fenet	LIRIS, University Lyon 1, France
Michele Lombardi	DISI, University of Bologna, Italy
Barry O'Sullivan	University College Cork, Ireland
Emmanuel Prados	Inria, France
Francesca Rossi	University of Padova, Italy
Louis-Martin Rousseau	École Polytechnique de Montreal, Canada
Helmut Simonis	Insight Centre for Data Analytics, Ireland
Christine Solnon	LIRIS, INSA Lyon, France
Charlotte Truchet	LINA, University de Nantes, France
Pascal Van Hentenryck	University of Michigan, USA
Willem-Jan Van Hoeve	Carnegie Mellon University, USA

CP and Biology Track Program Committee

Agostino Dovier	University of Udine, Italy
Alessandro Dal Palù	University of Parma, Italy
Rolf Backofen	Albert Ludwigs University of Freiburg, Germany
Pedro Barahona	Universidade Nova de Lisboa, Portugal
Alexander Bockmayr	Freie Universität Berlin, Germany
Mats Carlsson	SICS, Sweden
Simon De Givry	INRA – UBIA, France
François Fages	Inria Paris-Rocquencourt, France

Ines Lynce   INESC-ID/IST, University of Lisbon, Portugal
Enrico Pontelli   New Mexico State University, USA
Sebastian Will   University of Leipzig, Germany
Sylvain Soliman   Inria Paris-Rocquencourt, France
Nigel Martin   Birkbeck, University of London, UK
Alberto Policriti   University of Udine, Italy

Music Track Track Program Committee

Truchet Charlotte   University of Nantes, France
Assayag Gérard   IRCAM, France
Herremans Dorien   Queen Mary University of London, UK
Pearson Justin   Uppsala University, Sweden
Rueda Camilo   Universidad Javeriana, Colombia
Sandred Örjan   University of Manitoba, Canada

Preferences, Social Choice, and Optimization Track Committee

Haris Aziz   Data61/University of New South Wales, Australia
Fahiem Bacchus   University of Toronto, Canada
Peter Biró   Hungarian Academy of Sciences, Hungaria
Piotr Faliszewski   AGH University of Science and Technology, Poland
Umberto Grandi   University of Toulouse, France
Philip Kilby   NICTA and the Australian National University, Australia
Jerome Lang   University of Paris-Dauphine, France
David Manlove   University of Glasgow, Scotland, UK
Nick Mattei   Data61/NICTA and University of New South Wales,
                             Australia
Nicolas Maudet   Pierre et Marie Curie University, France
Amnon Meisels   Ben-Gurion University of the Negev, Israel
Nina Narodytska   Samsung Research America, USA
Maria Silvia Pini   University of Padova, Italy
Patrick Prosser   University of Glasgow, Scotland, UK
Jorg Rothe   University of Düsseldorf, Germany
Paul Shaw   IBM, France
Brent Venable   Tulane University, USA

Testing and Verification Track Committee

Andreas Podelski   University of Freiburg, Germany
Arnaud Gotlieb   Simula Research Laboratory, Norway
Catherine Dubois   ENSIIE-CEDRIC, France
Mats Carlsson   SICS, Sweden
Daniel Le Berre   CNRS - Université d'Artois, France
Nadjib Lazaar   UM2-LIRMM, France
Giorgio Delzanno   Università di Genova, Italy

Justin Pearson	Uppsala University, Sweden
Pierre Flener	Uppsala University, Sweden
Parosh Aziz Abdulla	Uppsala University, Sweden
Sebastien Bardin	CEA LIST, France
Harald Sondergaard	University of Melbourne, Australia
Peter J. Stuckey	University of Melbourne, Australia
Joxan Jaffar	National University of Singapore, Singapore
Eyal Bin	IBM, Israel
Cédric Pralet	ONERA, France
Justyna Petke	University College London, UK
Morten Mossige	ABB Robotics, Norway

Additional Reviewers

Özgür Akgün	Ian Gent	Margaux Nattaf
Martin Aleksandrov	Grigori German	Peter Nightingale
Goldsztejn Alexandre	Phillippe Grangier	Carlos Olarte
Noureddine Aribi	Gabriel Hjort Blindell	Alexandre Papadopoulos
Gilles Audemard	Vinasétan Ratheil Houndji	Anastasia Paparrizou
Rehan Abdul Aziz	Marie-José Huguet	Marie Pelleau
Johannes	Peter Jeavons	Quentin Plazar
Gerhardus Benade	Chris Jefferson	Gerhard Rauchecker
Nikolaj Bjorner	Elias Khalil	Camilo Rocha
Alessio Bonfietti	Ryo Kimura	Olivier Roussel
Nicolas Bonifas	Lars Kotthoff	Andrew Edward Santosa
Andrea Borghesi	Ludwig Krippahl	Andreas Schutt
Eric Bourreau	Arnaud Lallouet	Thiago Serra
Valentina Cacchiani	Ronan Le Bras	Mohamed Siala
Clément Carbonnel	Jan Leike	Sebastien Tabary
Roberto	Olivier Lhomme	Guido Tack
Castaneda Lozano	Xavier Lorca	Navid Talebanfard
Raffaele Cipriano	Samir Loudni	Miguel Terra-Neves
Andre Augusto Cire	Vasco Manquinho	Cyril Terrioux
Martin Cooper	Paolo Marin	Gilles Trombettoni
Allegra De Filippo	Ruben Martins	Sascha Van Cauwelaert
Cyrille Dejemeppe	Valentin	Wei Xia
Stefano Di Alesio	Mayer-Eichberger	Neng-Fa Zhou
Sylvain Ducomman	Ciaran McCreesh	Zichen Zhu
Ferdinando Fioretto	Michele Monaci	
Steven Gay	Pedro T. Monteiro	

Invited Talks

Horn Constraints for Software Verification and Synthesis

Andrey Rybalchenko

Microsoft Research
`rybal@microsoft.com`

Abstract. We will show how Horn constraints can be used to describe verification and synthesis problems, and how such constraints can be solved efficiently. In particular we will demonstrate how cardinality operators help to reason about quantitative properties and carry out counting-based correctness arguments, which are useful for the verification of information flow properties and parametrized systems.

Optimizing Preferences and Social Welfare in Healthcare-Related Matching Problems

David F. Manlove

School of Computing Science, Sir Alwyn Williams Building,
University of Glasgow, Glasgow G12 8QQ, UK
`david.manlove@glasgow.ac.uk`

Abstract. Matching problems typically involve assigning agents to commodities, possibly on the basis of ordinal preferences or other metrics. These problems have large-scale applications to centralized matching schemes in many countries and contexts. For example, such schemes are used for the annual allocation of junior doctors to hospitals in the USA, Canada and Japan, for higher education admission in China, Hungary and Turkey, and for placing military cadets to branches in the USA.

In this talk I will describe the matching problems featuring in two centralized schemes in the UK that have involved collaborations between the National Health Service and the University of Glasgow. One of these dealt with the allocation of junior doctors to Scottish hospitals (as part of the Scottish Foundation Allocation Scheme, running from 1999–2012), and the other is concerned with finding kidney exchanges among incompatible donor-patient pairs across the UK (under the auspices of the National Living Donor Kidney Sharing Schemes, in operation since 2007).

The case of junior doctor allocation can be modelled by the Hospitals/Residents problem, where we seek a *stable matching*. Although the classical problem in its simplest form is solvable in polynomial time, when couples apply jointly in pairs, and when preference lists may include ties, as could occur in the Scottish application, the problem of finding a stable matching becomes NP-hard.

For kidney exchange, the problem can be modelled via cycle packing in a directed graph, where cycles cannot exceed a given fixed length k. We seek a vertex-disjoint collection of cycles that covers as many vertices as possible, in order to maximise the number of potential transplants – again this problem is NP-hard even if $k = 3$, as in the UK application.

In each case I will describe the applications, present the underlying algorithmic problems, and outline how integer and constraint programming techniques have been used to tackle these NP-hard problems over the years. I will then give an overview of computational results arising from executions on real data connected with the associated matching schemes in recent years.

Evidence-Based Optimization of Complex Infrastructures

Pascal Van Hentenryck

University of Michigan
`pvanhent@umich.edu`

Abstract. For the first time in the history of humankind, we are accumulating data sets of unprecedented scale and accuracy about physical infrastructures, natural phenomena, man-made processes, and human behavior. These developments, together with progress in high-performance computing, predictive models, and operations research, offer novel opportunities for optimizing complex infrastructures holistically. We present some exciting projects in evidence-based optimization and highlight some challenges and opportunities for constraint programming in this space.

Optimization and Control in the Smart Grid and Beyond

Zico Kolter

School of Computer Science at Carnegie Mellon University
zkolter@cs.cmu.edu

Abstract. The world's electrical energy system is transforming, evolving from a "top-down" purely physics-driven process to a digital, interconnected system with bidirectional control. This new electrical grid, broadly referred to as the smart grid, offers many opportunities for advanced optimization and AI techniques to play a transformational role. This talk will highlight some general themes and optimization problems that arise frequently in these settings, and also discuss some of our work on general stochastic control approaches in these settings. I will then close by discussing some broad themes in general-purpose optimization, inspired by the smart grid setting, but with general applicability to a wide range of problems.

Contents

Application Track

Music Track

Preference, Social Choice and Optimization Track

Testing and Verification Track

Technical Track

Exploiting Short Supports for Improved Encoding of Arbitrary Constraints into SAT

Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel,
and Peter Nightingale[(⊠)]

School of Computer Science, University of St Andrews, St Andrews, UK
{ozgur.akgun,ian.gent,caj21,ijm,pwn1}@st-andrews.ac.uk

Abstract. Encoding to SAT and applying a highly efficient modern SAT solver is an increasingly popular method of solving finite-domain constraint problems. In this paper we study encodings of arbitrary constraints where unit propagation on the encoding provides strong reasoning. Specifically, unit propagation on the encoding simulates generalised arc consistency on the original constraint. To create compact and efficient encodings we use the concept of short support. Short support has been successfully applied to create efficient propagation algorithms for arbitrary constraints. A short support of a constraint is similar to a satisfying tuple however a short support is not required to assign every variable in scope. Some variables are left free to take any value. In some cases a short support representation is smaller than the table of satisfying tuples by an exponential factor. We present two encodings based on short supports and evaluate them on a set of benchmark problems, demonstrating a substantial improvement over the state of the art.

1 Introduction

We address the problem of encoding constraint problems into SAT. This is an important step because it allows us to leverage the rich modelling languages available in constraints such as MiniZinc [26] and Essence Prime [20]. We have previously shown that the constraint modelling tool SAVILE ROW [17] can be used to translate constraint problems directly to SAT, exploiting automated modelling techniques such as common subexpression elimination [21].We add to the important and growing literature on modelling of constraints in SAT [6]. Most study has been devoted to constraints such as linear constraints including the special case of cardinality constraints [1,2,8,24].

In this paper we show that we can improve SAT models of table constraints by exploiting short supports. Table constraints are vital in constraint modelling as they allow arbitrary constraints to be expressed. Table constraints can be expressed in SAT in such a way as to ensure that unit propagation in the SAT encoding performs reasoning equivalent to that done by generalised arc consistency (GAC) in the constraint problem [3]. A short support of a constraint is similar to a satisfying tuple, but a short support is not required to assign every

variable: some variables are left free to take any value. Where it is possible, exploiting short supports has proved to improve efficiency of GAC propagation [12,19]. We show that Bacchus's encoding of table constraints into SAT can be adapted to exploit short supports. This can lead to much smaller encodings and faster propagation, while still obtaining GAC. We present two encodings for table constraints with short supports into SAT. We get the advantages of modern SAT solvers automatically, such as generating explanations of failure and learning.

Short supports represent one method of compressing table constraints, and other methods have been proposed: MDDs [7], C-tuples [13] and their generalisation [22], and Smart Tables [14]. Uniquely, short supports allow us to directly improve the encoding of Bacchus without introducing any complications to it.

2 Preliminaries

The Propositional Satisfiability Problem (SAT) is to find an assignment to a set of Boolean variables so as to satisfy a given Boolean formula, typically expressed in conjunctive normal form [5]. SAT has many important applications, such as hardware design and verification, planning, and combinatorial design [15]. Powerful, robust solvers have been developed for SAT employing techniques such as conflict-driven learning, watched literals, restarts and dynamic heuristics for backtracking solvers [16], and sophisticated incomplete techniques such as stochastic local search [23].

A *constraint satisfaction problem* (CSP) is defined as a set of variables X, a function that maps each variable to its domain, $D : X \to 2^{\mathbb{Z}}$ where each domain is a finite set, and a set of constraints C. A constraint $c \in C$ is a relation over a subset of the variables X. The *scope* of a constraint c, named scope(c), is the set of variables that c constrains. During a systematic search for a solution to a CSP, values are progressively removed from the domains D. Therefore, we distinguish between the *initial* domains and the *current* domains. The function D refers to the current domains and D_s to the initial domains. A *literal* is a variable-value pair (written $x \mapsto v$). A literal $x \mapsto v$ is *valid* if $v \in D(x)$. The size of the largest initial domain is d. For a constraint c we use r for the size of scope(c). A constraint c is Generalised Arc Consistent (GAC) if and only if there exists a full-length support containing every valid literal of every variable in scope(c). GAC is established by identifying all literals $x \mapsto v$ for which no full-length support exists and removing v from the domain of x. We consider only algorithms for establishing GAC in this paper. A *full-length support* of constraint c is a set of literals containing exactly one literal for each variable in scope(c), such that c is satisfied by the assignment represented by these literals.

A *short support* is a support containing at most one literal for each variable in scope(c). As a motivating example for short supports, consider the lexicographic ordering constraint $\leq_{lex}$ on tuples. We can often know this is true just based on examining a small number of the variables in the constraint. Consider $\langle x_1, x_2, x_3 \rangle \leq_{lex} \langle x_4, x_5, x_6 \rangle$ where variables $x_1, \dots x_6$ each have initial domain $D_s = \{1, 2, 3\}$. A full-length support is necessarily size 6: e.g.

$\{x_1 \mapsto 2, x_2 \mapsto 2, x_2 \mapsto 2, x_4 \mapsto 2, x_5 \mapsto 2, x_6 \mapsto 2\}$ is a correct full-length support since $\langle 2, 2, 2 \rangle \leq_{lex} \langle 2, 2, 2 \rangle$. In contrast, the set $\{x_1 \mapsto 1, x_4 \mapsto 2\}$ is a correct short support even though it contains only two of the six variables: it is necessarily true that $\langle 1, *, * \rangle \leq_{lex} \langle 2, *, * \rangle$, whatever replaces the stars. Short supports can be of variable lengths as needed. For example, the short support $\{x_1 \mapsto 2, x_2 \mapsto 2, x_4 \mapsto 2, x_5 \mapsto 3\}$ is a correct short support of size 4, but no literal can be removed from it without leaving at least one extension to a set of literals breaking the constraint. Following [19] we formally define short support as follows.

Definition 1 *[Short support]. A **short support** S for constraint c and domains D_s is a set of literals $x \mapsto v$ such that $x \in \text{scope}(c)$, $x \mapsto v$ is valid w.r.t D_s, x occurs only once in S, and every superset of S that contains one valid (w.r.t D_s) literal for each variable in $\text{scope}(c)$ is a full-length support.*[1]

Note from the definition that any full-length support is also a short support. In the example the set $\{x_1 \mapsto 2, x_2 \mapsto 2, x_2 \mapsto 2, x_4 \mapsto 2, x_5 \mapsto 2, x_6 \mapsto 2\}$ is a short support and indeed no literal can be omitted to give another short support. In some cases even an empty set can be a short support. Suppose we change the motivating example so that $D_s(x_1) = \{0\}$ and other domains are unchanged. All valid assignments satisfy the lexicographic constraint since the only value of x_1 is 0 and $\langle 0, *, * \rangle \leq_{lex} \langle *, *, * \rangle$, so the empty set is a correct short support.

3   Encoding Table Constraints into SAT

Our encoding of constraint problems into SAT follows that which we have previously used and reported on [21]. When encoding a CSP variable, SAVILE ROW provides SAT literals for facts about the variable: $[x = a]$, $[x \neq a]$, $[x \leq a]$ and $[x > a]$ for a CSP variable x and value a. On all benchmarks used here, CSP variables are encoded in two ways. A variable with domain size 2 is represented with a single SAT variable. For variables with larger domains we have one SAT variable representing $[x = a]$ for each value $a \in D_s(x)$, and one SAT variable for each $[x \leq a]$ except $[x \leq \max(D_s(x))]$ that would always be true. Also, $[x = \max(D_s(x))] \leftrightarrow \neg[x \leq \max(D_s(x)) - 1]$ saving one more SAT variable. If we have a literal, e.g. $[x \leq a]$, where $a \notin D_s(x)$, then the literal is mapped as appropriate to True, False or an equivalent literal, e.g. $[x \leq b]$ for $b = \max(\{i \in D_s(x) | i < a\})$. The encoding has $2|D_s(x)| - 2$ SAT variables and consistency among them is maintained by the following clause set (sometimes called the ladder encoding [10]).

$$\forall a \in D_s(x). \quad [x = a] \leftrightarrow ([x \leq a] \wedge \neg[x \leq a + 1]) \quad \wedge \quad [x \leq a - 1] \rightarrow [x \leq a]$$

---

[1] The set of short supports depends on the domains D_s. We always use the initial domains. Elsewhere, short supports are generated using the current domains D but these sets are not necessarily short supports after backtracking [18,19]. A support of either type is valid iff all literals in it are valid.

The only constraint we consider in this paper is the table constraint. This can be used to encode arbitrary constraints extensionally. The table constraint is very important in constraint programming, for constraints where no convenient expression in terms of simpler constraints is available. Suppose we have a constraint C on variables $x_1 \ldots x_r$ represented as a table of satisfying tuples, each of which is valid w.r.t. initial domains. Bacchus presented an encoding of table constraints [3]. Each satisfying tuple τ_i (where $i \in \{1 \ldots m\}$) is represented with an auxiliary SAT variable t_i. The first clause set ensures that each t_i becomes *false* when the tuple τ_i becomes invalid (i.e. a value in τ_i has been removed).

$$\forall i \in \{1 \ldots m\}. \quad \forall j \in \{1 \ldots r\}. \quad ([x_j = \tau_i[j]] \vee \neg t_i)$$

The second clause set states that each domain value of variables $x_1 \ldots x_r$ must be supported by a valid tuple.

$$\forall i \in \{1 \ldots r\}. \quad \forall a \in D(x_i). \quad ([x_i \neq a] \vee \bigvee t_j \quad \text{where} \quad \tau_j[i] = a)$$

Unit propagation (UP) applied to these clauses firstly removes from consideration any invalid tuple τ_i by setting t_i to *false*, then removes any domain value a of variable x_i (by setting $[x_i \neq a]$) where no remaining tuples support the value. Thus UP (re-)establishes GAC. Bacchus observed that the encoding has size $O(mr)$ which is linear in the size of the table representation of the constraint and applying UP has the same time complexity as a generic GAC propagator.

4   Short Support Encodings of Arbitrary Constraints

The idea of short support has already been successfully applied in constraint propagators [12,18,19]. Short support is defined above (Definition 1). Our contribution here is to exploit short supports in a new encoding that is smaller and more efficient than the Bacchus encoding while keeping the property that unit propagation establishes GAC. We assume that we already have a short support set for the constraint we wish to encode. It is often straightforward to construct a short support set for a constraint. Otherwise, an automated approach may be used such as the Greedy-Compress algorithm [12] that takes the table of full-length supporting tuples and compresses them to a short support set. It is possible that no short supports are available: in this case our encoding will provide neither harm nor benefit as it is equivalent to the Bacchus encoding.

The encoding for constraint C on variables $x_1 \ldots x_r$ is as follows. Each short support σ_i ($i \in \{1 \ldots m\}$) is represented with an auxiliary SAT variable s_i. The first clause set ensures that s_i is *false* when σ_i contains a literal that is invalid.

$$\forall i \in \{1 \ldots m\}. \quad \forall (x_j \mapsto a) \in \sigma_i. \quad ([x_j = a] \vee \neg s_i)$$

The second clause set states that each literal $(x_i \mapsto a)$ of variables $x_1 \ldots x_r$ must be supported by a valid short support, either *explicitly* (where the short support simply contains $(x_i \mapsto a)$) or *implicitly* (where the short support contains no literal of the variable x_i, meaning x_i may take any value).

$$\forall i \in \{1 \ldots r\}. \; \forall a \in D_s(x_i).$$
$$([x_i \neq a] \vee \bigvee s_j \text{ where } (x_i \mapsto a) \in \sigma_j \text{ or } \forall b.(x_i \mapsto b) \notin \sigma_j)$$

The two clause sets are sufficient for unit propagation to establish GAC on the constraint: the first clause set removes from consideration any short support that is invalid by setting the relevant s_i to *false*, and the second prunes any values that have no remaining short supports of either type (explicit or implicit).

This encoding has the property that the auxiliary variables s_i may not be uniquely determined when all SAT variables representing CSP variables have been assigned. This occurs when more than one short support is valid w.r.t. the CSP assignment. In this case at least one of the corresponding s_i must be true, but otherwise their values float freely. The free variables may cause additional search, and would cause a problem if we wished to count solutions. We obtain a second encoding without this issue by including the following additional clause set, which sets an s_i variable to true when all literals in σ_i are set true.

$$\forall i \in \{1 \ldots m\}. \quad \left(\bigvee [x_j \neq a] \quad \text{where} \quad (x_j \mapsto a) \in \sigma_i \right) \vee s_i$$

Compared to the full-length table encoding, the short support encoding has fewer auxiliary variables, each representing a smaller conjunction of literals of the primary SAT variables. This is likely to be beneficial for conflict learning, facilitating more general and reusable explanations for conflicts.

We will refer to the short support encoding without the optional clause set as ShortTableSAT, and with the optional clause set as ShortTableSAT+.

5   Experimental Evaluation

To show the potential benefit of encoding using short supports, we evaluated our encodings of them on a number of problem classes. These are not intended to be an exhaustive or representative sample of possible problems, but a set of instances where short supports are available and thus show the potential benefit of our encodings. The instances we study are drawn from three general categories.

5.1   Case Study 1: Rectangle Packing

The rectangle packing problem [25] (with parameters n, *width* and *height*) consists of packing all squares from size 1×1 to $n \times n$ into the rectangle of size *width* $\times$ *height*. This is modelled as follows: we have variables $x_1 \ldots x_n$ and $y_1 \ldots y_n$, where (x_i, y_i) represents the Cartesian coordinates of the lower-left corner of the $i \times i$ square. Domains of x_i variables are $\{0 \ldots width - i\}$, and for y_i variables are $\{0 \ldots height - i\}$. The only type of constraint is non-overlap of squares i and j: $(x_i + i \leq x_j) \vee (x_j + j \leq x_i) \vee (y_i + i \leq y_j) \vee (y_j + j \leq y_i)$. The domains of x_n and y_n are reduced to break flip symmetries [25]. The short supports of the non-overlap constraints are all of length two. Each short support satisfies one of the four disjuncts, thus satisfying the constraint. In a given instance, each constraint has a distinct short supports table because the constants i and j differ.

We compared the full length table encoding to both short table encodings on a set of instances taken from Jefferson and Nightingale [12] in addition to some generated ones. We generated two sets of instances: some small instances for all combinations of values for $n \in \{2 \ldots 6\}$, $width \in \{10, 15, 20, 25\}$, and $height \in \{10, 15, 20, 25\}$; and some larger instances for all combinations of values for $n \in \{2, 4 \ldots 30\}$, $width \in \{20, 25, 30, 35\}$, and $height \in \{20, 25, 30, 35\}$. For both of these sets we only kept those instances where $width < height$ and also filtered out those which were trivially unsatisfiable due to area constraints.

5.2   Case Study 2: The Oscillating Life Problem and Variants Thereof

We consider the problem of maximum density oscillators (repeating patterns) in John Conway's Game of Life. We consider this and three variants. Immigration has two *alive* states. When a cell becomes alive, it takes the state of the majority of the 3 neighbouring live cells that caused it to become alive. Otherwise the rules of Immigration are the same as those of Life. Quadlife has four *alive* states. When a cell becomes alive, it takes the state of the majority of the 3 neighbouring live cells which caused it to become alive, unless all 3 neighbours have different colours in which case it takes the colour which none of its neighbours have. Apart from this the rules are the same as Life. Finally Brian's Brain has three states: *dead*, *alive* and *dying*. If a cell is *dead* and has exactly two *alive* (not dying) neighbours, it will become *alive*, otherwise it remains *dead*. If a cell is *alive*, it becomes *dying* after one time step. If a cell is *dying*, it becomes *dead* after one time step. We use an $n \times n$ grid with t time steps, for all pairs of values (n, t) where $n \in \{3 \ldots 7\}$ and $t \in \{2 \ldots 6\}$, giving 25 instances.

We use the problem and constraint model as described by Gent et al. [9]. For all four problems, we make one change: we minimise the occurrences of the value 0 (dead) in all layers. For Immigration, Quadlife and Brian's Brain we also add extra domain values for each additional state. For each cell and each time step, a single constraint links the cell and its eight neighbours to the same cell in the next time step. Therefore the constraints have arity 10. Short supports arise from sums in the rules, e.g. a live cell with more than three live neighbours will die: if the current cell is alive, any four neighbours are alive, and the next cell is dead then the constraint is satisfied and we have a short support of length 6.

5.3   Case Study 3: The Antichain Problem

The antichain problem is to find a set of multisets under some conditions [11]. Representing a multiset as a vector of integers (giving the cardinality of each possible value), we find a set of size n of vectors of length l, containing integers from the set $\{0 \ldots d-1\}$. For each pair of vectors v_1, v_2, there must exist an index i where $v_1[i] < v_2[i]$ and a second index j where $v_1[j] > v_2[j]$. The problem is modelled with a two-dimensional matrix A with size n by l. Each pair of vectors is linked by a single constraint capturing both the $<$ and $>$ requirements, with scope size $2l$. The constraint linking any two vectors has short supports of length four. We compared the full length table encoding to both short table encodings on a set of 50 instances that includes all instances used by Jefferson et al. [11].
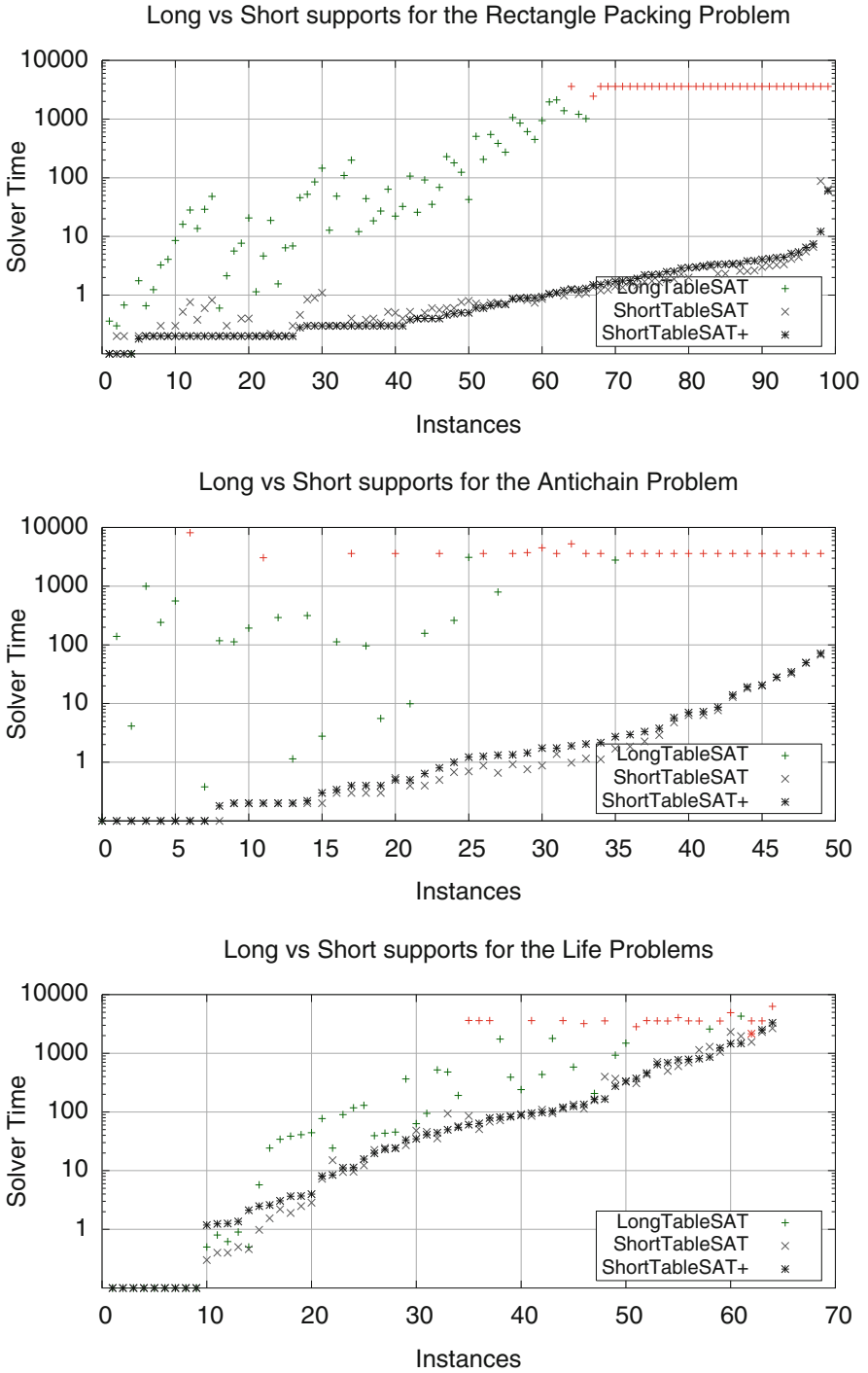
Fig. 1. Solver run times for the six problem classes

5.4 Experimental Results

For a SAT solver we used the SAT'14 Competition version of Lingeling [4] (version ayv 86bf266b9332599f1b876e28a02fe8427aeaa2db). Each instance was solved 5 times, with random seeds changing from 1 to 5. We report the median of the five runtimes reported by Lingeling. Experiments were performed with 32 processes in parallel on a 32-core AMD Opteron 6272 at 2.1 GHz with 256 GB RAM. We set a limit of 1 h for each Lingeling process. Results are in Fig. 1. The x axis shows instances, ordered by increasing run time of ShortTableSAT+. Each position on the x axis represents the same instance within a plot. The y axis shows run time in seconds of Lingeling. Run time of the SAVILE ROW translation process is ignored, except that nothing is plotted if SAVILE ROW overran its time or space limit. We do plot (in red) the points where Lingeling reported that it reached its time limit. In some cases Lingeling reported reaching the time limit but also reported a time substantially less or greater than 1 h. We simply plotted the time Lingeling reported using a red point.

For the packing problem (Fig. 1, top), we see that we benefit greatly from use of short supports. There are many instances where LongTableSAT is unable to solve the instance, but both short table encodings are. On most other instances both short support methods are at least one and often several orders of magnitude faster. There is no clear preference between the two short encodings, with both methods faster on some instances, although ShortTableSAT+ is typically faster on the instances which can be solved fastest. We see in the antichain problem (Fig. 1, middle) that again short supports provide improved search performance compared to LongTableSAT, by orders of magnitude. In this case it seems that ShortTableSAT is the better of the two short table encodings. For the Life, Immigration, Brian's Brain and Quadlife problem classes (Fig. 1, bottom), we see that short supports do improve search but to a much lesser degree than in the previous cases. There are a small number of cases where LongTableSAT beats one of the short methods. However, using short tables is still much faster in most cases.

Our results show that the use of short supports in a SAT encoding can greatly improve solving performance over the use of full length table constraints.

6 Conclusions

Encoding to SAT and solving with a modern CDCL SAT solver is a very effective way to solve difficult finite-domain constraint problems. We have studied the encoding of table constraints, and proposed two new encodings based on the idea of *short supports*. These improve upon an existing encoding in both size and solving efficiency. In our experiments, the new encoding is consistently faster, frequently by over 10 times and in some cases by over 1000 times.

References

1. Abío, I., Mayer-Eichberger, V., Stuckey, P.J.: Encoding linear constraints with implication chains to CNF. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 3–11. Springer, Heidelberg (2015). doi:10.1007/978-3-319-23219-5_1

2. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks: a theoretical and empirical study. Constraints **16**(2), 195–221 (2011). doi:10.1007/s10601-010-9105-0

3. Bacchus, F.: GAC via unit propagation. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 133–147. Springer, Heidelberg (2007)

4. Biere, A.: Lingeling, plingeling and treengeling entering the sat competition 2013. In: Proceedings of SAT Competition, pp. 51–52 (2013)

5. Biere, A., Heule, M., van Maaren, H.: Handbook of Satisfiability, vol. 185. IOS Press, Amsterdam (2009)

6. Brain, M., Hadarean, L., Kroening, D., Martins, R.: Automatic generation of propagation complete SAT encodings. In: Jobstmann, B., Leino, K.R.M. (eds.) VMCAI 2016. LNCS, vol. 9583, pp. 536–556. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49122-5_26

7. Cheng, K.C.K., Yap, R.H.C.: An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. Constraints **15**(2), 265–304 (2010)

8. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. JSAT **2**(1–4), 1–26 (2006). http://jsat.ewi.tudelft.nl/content/volume2/JSAT2_1_Een.pdf

9. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Generating special-purpose stateless propagators for arbitrary constraints. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 206–220. Springer, Heidelberg (2010)

10. Gent, I.P., Nightingale, P.: A new encoding of alldifferent into SAT. In: Proceedings 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems (ModRef 2004), pp. 95–110 (2004)

11. Jefferson, C., Moore, N., Nightingale, P., Petrie, K.E.: Implementing logical connectives in constraint programming. Artif. Intell. **174**, 1407–1429 (2010)

12. Jefferson, C., Nightingale, P.: Extending simple tabular reduction with short supports. In: Proceedings of 23rd International Joint Conference on Artificial Intelligence (IJCAI), pp. 573–579 (2013)

13. Katsirelos, G., Walsh, T.: A compression algorithm for large arity extensional constraints. In: Proceedings of the CP 2007, pp. 379–393 (2007)

14. Mairy, J.-B., Deville, Y., Lecoutre, C.: The smart table constraint. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 271–287. Springer, Heidelberg (2015)

15. Marques-Silva, J.: Practical applications of boolean satisfiability. In: 9th International Workshop on Discrete Event Systems (WODES 2008), pp. 74–80 (2008)

16. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Annual Design Automation Conference, pp. 530–535. ACM (2001)

17. Nightingale, P., Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I.: Automatically improving constraint models in savile row through associative-commutative common subexpression elimination. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 590–605. Springer, Heidelberg (2014)

18. Nightingale, P., Gent, I.P., Jefferson, C., Miguel, I.: Exploiting short supports for generalised arc consistency for arbitrary constraints. In: Proceedings of the IJCAI 2011, pp. 623–628 (2011)

19. Nightingale, P., Gent, I.P., Jefferson, C., Miguel, I.: Short and long supports for constraint propagation. J. Artif. Intell. Res. **46**, 1–45 (2013)
20. Nightingale, P., Rendl, A.: Essence' description (2016). arXiv:1601.02865 [cs.AI]
21. Nightingale, P., Spracklen, P., Miguel, I.: Automatically improving SAT encoding of constraint problems through common subexpression elimination in savile row. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 330–340. Springer, Heidelberg (2015)
22. Régin, J.: Improving the expressiveness of table constraints. In: The 10th International Workshop on Constraint Modelling and Reformulation (ModRef 2011) (2011)
23. Shang, Y., Wah, B.W.: A discrete Lagrangian-based global-search method for solving satisfiability problems. J. Glob. Optimi. **12**(1), 61–99 (1998)
24. Marques-Silva, J., Lynce, I.: Towards robust CNF encodings of cardinality constraints. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 483–497. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74970-7_35
25. Simonis, H., O'Sullivan, B.: Search strategies for rectangle packing. In: Proceedings of the CP 2008, pp. 52–66 (2008)
26. Stuckey, P.J., Tack, G.: MiniZinc with functions. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013. LNCS, vol. 7874, pp. 268–283. Springer, Heidelberg (2013)

Systematic Derivation of Bounds and Glue Constraints for Time-Series Constraints

Ekaterina Arafailova[1]([✉]), Nicolas Beldiceanu[1], Mats Carlsson[2], Pierre Flener[3],
María Andreína Francisco Rodríguez[3], Justin Pearson[3], and Helmut Simonis[4]

[1] TASC (CNRS/Inria), Mines Nantes, 44307 Nantes, France
{Ekaterina.Arafailova,Nicolas.Beldiceanu}@mines-nantes.fr
[2] SICS, P.O. Box 1263, 164 29 Kista, Sweden
Mats.Carlsson@sics.se
[3] Department of Information Technology, Uppsala University,
751 05 Uppsala, Sweden
{Pierre.Flener,Maria.Andreina.Francisco,Justin.Pearson}@it.uu.se
[4] Insight Centre for Data Analytics, University College Cork, Cork, Ireland
Helmut.Simonis@insight-centre.org

Abstract. Integer time series are often subject to constraints on the aggregation of the integer features of all occurrences of some pattern within the series. For example, the number of inflexions may be constrained, or the sum of the peak maxima, or the minimum of the peak widths. It is currently unknown how to maintain domain consistency efficiently on such constraints. We propose parametric ways of systematically deriving glue constraints, which are a particular kind of implied constraints, as well as aggregation bounds that can be added to the decomposition of time-series constraints [5]. We evaluate the beneficial propagation impact of the derived implied constraints and bounds, both alone and together.

1 Introduction

A *time series* is here a sequence of integers, corresponding to measurements taken over a time interval. Time series are common in many application areas, such as the output of electric power stations over multiple days [8], or the manpower required in a call-centre [3].

We showed in [5] that many constraints $\gamma(\langle X_1, \ldots, X_n \rangle, N)$ on an unknown time series $X = \langle X_1, \ldots, X_n \rangle$ of given length n can be specified by a triple $\langle \sigma, f, g \rangle$, where σ is a regular expression over the alphabet $\Sigma = \{'<', '=', '>'\}$

(we assume the reader is familiar with regular expressions and automata [12]), while $f \in \{\texttt{max}, \texttt{min}, \texttt{one}, \texttt{surface}, \texttt{width}\}$ is called a *feature*, and $g \in \{\texttt{Max}, \texttt{Min}, \texttt{Sum}\}$ is called an *aggregator*. Let the sequence $S = \langle S_1, \ldots, S_{n-1} \rangle$, called the *signature* and containing *signature variables*, be linked to X via the *signature constraints* $(X_i < X_{i+1} \Leftrightarrow S_i = \text{'<'}) \wedge (X_i = X_{i+1} \Leftrightarrow S_i = \text{'='}) \wedge (X_i > X_{i+1} \Leftrightarrow S_i = \text{'>'})$ for all $i \in [1, n-1]$. A σ-*pattern* is a sub-series of X that *corresponds to* a maximal occurrence of σ within S. Integer variable N is constrained to be the aggregation, computed using g, of the list of values of feature f for all σ-patterns in X. A set of 20 regular expressions is considered. We name a time-series constraint specified by $\langle \sigma, f, g \rangle$ as g_f_σ.

Example 1. The time series $X = \langle 4, 4, 0, 0, 2, 4, 4, 7, 4, 0, 0, 2, 2, 2, 2, 2, 2, 0 \rangle$ has the signature $S = \text{'=>=<<=<>>=<====>'}$. Consider the regular expression $\texttt{Peak} = \text{'<(<|=)*(>|=)*>'}$: a $\texttt{Peak}$-pattern, called a *peak*, within a time series corresponds, except for its first and last elements, to a maximal occurrence of $\texttt{Peak}$ in the signature, and the $\texttt{width}$ feature value of a peak is its number of elements. The time series X contains two peaks, namely $\langle 2, 4, 4, 7, 4 \rangle$ and $\langle 2, 2, 2, 2, 2, 2 \rangle$, visible the way X is plotted in Fig. 1, of widths 5 and 6 respectively, hence the minimal-width peak, obtained by using the aggregator $\texttt{Min}$, has width $N = 5$: the underlying constraint is named MIN_WIDTH_PEAK. □
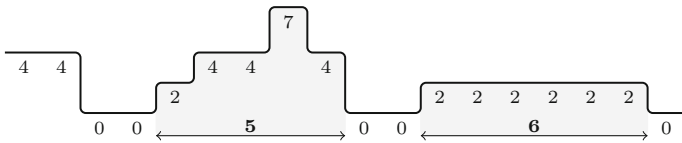
Fig. 1. MIN_WIDTH_PEAK$(5, \langle 4, 4, 0, 0, 2, 4, 4, 7, 4, 0, 0, 2, 2, 2, 2, 2, 2, 0 \rangle)$

After recalling in Sect. 2 further required background material on time-series constraints $g\_f\_\sigma(\langle X_1, \ldots, X_n \rangle, N)$, the contributions of this paper are ways of systematically deriving parametric implied constraints and bounds:

- We show in Sect. 3 how to derive systematically implied constraints, parameterised by aggregator g and feature f, for any regular expression σ.
- We give in Sect. 4 a methodology for systematically deriving bounds, parametrised by σ, on the variable N, for any pair of g and f, and then we demonstrate our methodology on the case when $g = \texttt{Max}$ and $f = \texttt{min}$.
- We evaluate in Sect. 5 the beneficial propagation impact of the derived implied constraints and bounds, both alone and together.

In Sect. 6, we conclude and discuss other related work. The implied constraints and bounds for all time-series constraints are in [2].

2   Background: Automata for Time-Series Constraints

In [5], we showed how to synthesise a deterministic finite automaton, enriched with accumulators [7], from any triple $\langle \sigma, f, g \rangle$ that specifies a time-series constraint. We now discuss the required background concepts using an example, namely the regular expression $\texttt{Peak} = \text{'}\texttt{<}(\texttt{<}|\texttt{=})\texttt{*}(\texttt{>}|\texttt{=})\texttt{*}\texttt{>}\text{'}$ of Example 1.
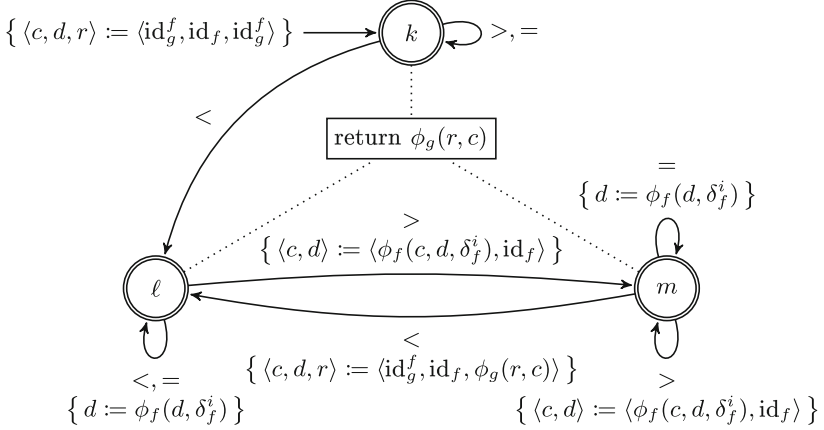
Fig. 2. Synthesised automaton for any $g\_f\_\textsc{Peak}$ constraint

The synthesised automaton for any $g\_f\_\textsc{peak}$ constraint is in Fig. 2. It returns the aggregation, using g, of the values of feature f for all $\texttt{Peak}$-patterns corresponding to the occurrences of $\texttt{Peak}$ within an input word over the alphabet $\Sigma = \{\text{'}\texttt{<}\text{'}, \text{'}\texttt{=}\text{'}, \text{'}\texttt{>}\text{'}\}$. The start state is k, annotated within braces by the initialisation of three accumulators: at any moment, accumulator c stores the feature value of the *current* $\texttt{Peak}$-pattern while d stores the feature value of a *potential* part of a $\texttt{Peak}$-pattern, and r stores the aggregated *result* for the feature values of the already encountered $\texttt{Peak}$-patterns. A transition is depicted by an arrow between two states and is annotated by a consumed alphabet symbol and, within braces, an accumulator update. The constants and operators appearing in the accumulator initialisation and updates are listed in Table 1; the binary operators ϕ_f and ϕ_g are used with arbitrary arity throughout this paper, in order to reduce the amount of parentheses. All states are accepting, an accepting state being marked by a double circle. Hence this automaton accepts the language Σ^*, but accepted words may be distinguished by the value of the returned expression, given within a box linked to all states. Note that the size of this automaton does *not* depend on the length of the input word.

In [7], we showed how to use an automaton with accumulators in order to decompose a constraint such as $g\_f\_\textsc{peak}(\langle X_1, \ldots, X_n \rangle, N)$ into signature constraints, linking $\langle X_1, \ldots, X_n \rangle$ to introduced signature variables $\langle S_1, \ldots, S_{n-1} \rangle$, as well as arithmetic and \textsc{table} constraints, linking $\langle S_1, \ldots, S_{n-1} \rangle$ and N to

Table 1. (Left) Features: identity, minimum, and maximum values; operators ϕ_f and δ_f^i recursively define the feature value v_u of a time series $\langle X_\ell, \ldots, X_u \rangle$ by $v_\ell = \phi_f(\mathrm{id}_f, \delta_f^\ell)$ and $v_i = \phi_f(v_{i-1}, \delta_f^i)$ for $i > \ell$, where δ_f^i is the contribution of X_i to v_u. (Right) Aggregators: operators and identity values relative to a feature f.

Feature f	id_f	$\min_f$	$\max_f$	ϕ_f	δ_f^i
one	1	1	1	1	1
width	0	0	$+\infty$	$+$	1
surface	0	$-\infty$	$+\infty$	$+$	X_i
max	$-\infty$	$-\infty$	$+\infty$	$\max$	X_i
min	$+\infty$	$-\infty$	$+\infty$	$\min$	X_i

Aggregator g	ϕ_g	id_g^f
Max	$\max$	$\min_f$
Min	$\min$	$\max_f$
Sum	$+$	0

introduced state variables Q_i and tuples $\langle C_i, D_i, R_i \rangle$ of accumulator variables, respectively denoting the automaton state and accumulator values $\langle c, d, r \rangle$ after consuming S_i. It is still unknown how to maintain domain consistency efficiently in general on this decomposition (see [7] for an analysis), hence implied constraints can help achieve more propagation, as we already showed in [6,11].

3   Glue Constraints for Time-Series Constraints

In [6] we derived an implied constraint, called a *glue constraint*, that can be added to the decomposition of a constraint specified by an automaton with accumulators: the derivation was *ad hoc* in most cases. In this paper, we introduce *parametric glue constraints* and show that they can be derived *automatically* for time-series constraints, which we introduced a year later in [5].

Example 2. We can explain the key insight using Example 1. The reverse of its time series X is $X' = \langle 0, 2, 2, 2, 2, 2, 2, 0, 0, 4, 7, 4, 4, 2, 0, 0, 4, 4 \rangle$ and has the signature $S^{\mathrm{mir}} = $ '$<=====>=<<>=>>=<=$', which we will call the *mirror* of the original signature S. The automaton of Fig. 2 returns the same value whether it consumes a signature or its mirror: the peaks of X are the reverses of the peaks of X' and the aggregation of their feature values is the same because all the operators ϕ_f and ϕ_g are commutative. We have this property for 19 of the 20 regular expressions in [5]. The idea now is to derive an implied constraint, which we will call a *glue constraint*, between the three accumulator triples of such an automaton after it has consumed (i) a signature w, (ii) a prefix w_1 of w, and (iii) the mirror of the corresponding suffix w_2 of w. For instance, let us split S into the prefix $P = $ '$=>=<<=<$' and the suffix $T = $ '$>>=<=====>$', which has the mirror $T^{\mathrm{mir}} = $ '$<=====>=<<$'. If we instantiate the automaton $\mathcal{A}$ of Fig. 2 for the MIN_WIDTH_PEAK constraint, that is with $f = \mathtt{width}$ and $g = \mathtt{Min}$, then $\mathcal{A}$ has the accumulator triples $\langle c, d, r \rangle = \langle 6, 0, 5 \rangle$ after consuming S, and $\langle c_1, d_1, r_1 \rangle = \langle +\infty, 3, +\infty \rangle$ after consuming P, and $\langle c_2, d_2, r_2 \rangle = \langle +\infty, 1, 6 \rangle$ after consuming T^{mir}. The value $\phi_g(r, c) = \min(5, 6) = 5$ returned by $\mathcal{A}$ on S can also be computed using the

formula $\phi_g(r_1, r_2, \phi_f(d_1, d_2, \delta_f^i))$, that is $\min(+\infty, 6, 3 + 1 + 1)$. That formula computes the minimum width of the following three peaks:

- the minimum-width peak corresponding to P, which actually has *no* occurrence of Peak = '$<(<|=)^*(>|=)^*>$', hence $r_1 = \mathrm{id}_f = +\infty$;
- the minimum-width peak corresponding to T^{mir}, whose only occurrence of Peak gives width $r_2 = 6$;
- the peak that is *created* by concatenating the following two potential peaks:
  - the potential occurrence of Peak at the end of P, giving width $d_1 = 3$;
  - the potential occurrence of Peak at the end of T^{mir}, giving $d_2 = 1$; note that if we feed T rather than T^{mir} to $\mathcal{A}$, then $\langle c_2, d_2, r_2 \rangle = \langle 6, 0, +\infty \rangle$ and d_2 reflects information about the *end* of T, rather than its beginning, hence the created peak is missed;

  but the contribution $\delta_f^i = 1$ (with $i = |P| + 1$) is required to compensate for the fact that $d_1 + d_2 = 4$ under-measures the width 5 of the created peak. □

We now formalise this insight, and add scenarios other than creation.

Definition 1 (mirror). *The* mirror *of a language L over $\Sigma = \{\text{'}<\text{'}, \text{'}=\text{'}, \text{'}>\text{'}\}$, denoted by L^{mir}, consists of the mirrors of all the words in L, where the* mirror *of a word or regular expression has the reverse order of its symbols and has all occurrences of the symbol '$<$' flipped into '$>$' and vice versa.*

We denote by $\mathcal{L}(\sigma)$ the regular language defined by a regular expression σ.

Definition 2 (state language). *Let q be a state of an automaton $\mathcal{A}$. The language accepted by q, denoted by $\mathcal{L}_q$, is the regular language accepted when q is made to be the only accepting state of $\mathcal{A}$.*

Example 3 Consider the automaton in Fig. 2. We have $\mathcal{L}_k = \mathcal{L}((>|=)^*)$, $\mathcal{L}_\ell = \Sigma^* \mathcal{L}(<(<|=)^*)$, and $\mathcal{L}_m = \Sigma^* \mathcal{L}(\text{Peak})\mathcal{L}(=^*)$, where Peak = '$<(<|=)^*(>|=)^*>$' is the regular expression for peaks. Standard algorithms of automata theory [12] can be used to compute state languages: we use the FAdo tool [1] to do so, as well as to check the language equalities stated in the following three examples. □

We concatenate two words by writing them side by side, with an implicit infix concatenation operator between them. The concatenation $L_1 L_2$ of two languages L_1 and L_2 is the language of all words $w_1 w_2$ where $w_1 \in L_1$ and $w_2 \in L_2$.

Definition 3 (extension). *We say that the concatenation $L_1 L_2$ extends a regular expression σ if and only if for any non-empty words $w_1 \in L_1$ and $w_2 \in L_2$ there exist a non-empty suffix s of w_1 and a non-empty prefix p of w_2 such that $sp \in \mathcal{L}(\sigma)$ and either s starts with the last occurrence of σ in w_1, where we say that $L_1 L_2$ extends the last σ in L_1, or p ends with the first occurrence of σ in w_2, where we say that $L_1 L_2$ extends the first σ in L_2, or both.*

Example 4. Consider the regular expression Peak = '$<(<|=)^*(>|=)^*>$'. Every word w_1 in $L_1 = \Sigma^* \mathcal{L}(\text{Peak})\mathcal{L}(=^*)$ has a suffix in $\mathcal{L}(\text{Peak})\mathcal{L}(=^*)$. Every word w_2 in $L_2 = \mathcal{L}((>|=)^*>)\Sigma^*$ has a prefix p in $\mathcal{L}((>|=)^*>)$. The concatenation sp is

in $\mathcal{L}(\texttt{Peak})\mathcal{L}(=^*)\mathcal{L}((>|=)^*>)$, which is a subset of $\mathcal{L}(\texttt{Peak})$, hence $L_1 L_2$ extends the last $\texttt{Peak}$ in L_1. Note that p cannot end with any occurrence of $\texttt{Peak}$, hence $L_1 L_2$ does not extend any $\texttt{Peak}$ in L_2.   □

Definition 4 (creation). *We say that the concatenation $L_1 L_2$ creates a regular expression σ if and only if for any non-empty words $w_1 \in L_1$ and $w_2 \in L_2$, there exist a non-empty suffix s of w_1 and a non-empty prefix p of w_2 such that $sp \in \mathcal{L}(\sigma)$ but neither does s start with an occurrence of σ in w_1 nor does p end with an occurrence of σ in w_2.*

Example 5. Consider again the regular expression $\texttt{Peak} = \text{'}<(<|=)^*(>|=)^*>\text{'}$. Every word w_3 in $L_3 = \Sigma^* \mathcal{L}(<(<|=)^*)$, such as P of Example 2, has a suffix in $\mathcal{L}(<(<|=)^*)$. Every word w_4 in $L_4 = \mathcal{L}((>|=)^*>)\Sigma^*$, such as T^{mir} of Example 2, has a prefix in $\mathcal{L}((>|=)^*>)$. The concatenation is in $\mathcal{L}(<(<|=)^*(>|=)^*>)$, which is equal to $\mathcal{L}(\texttt{Peak})$. However, neither can start with an occurrence of $\texttt{Peak}$ nor can end with an occurrence of $\texttt{Peak}$: hence $L_3 L_4$ does not extend $\texttt{Peak}$, but instead creates $\texttt{Peak}$.   □

We now give the glue constraint for a time-series constraint specified by $\langle \sigma, f, g \rangle$: it is specific to regular expression σ but generic in f and g. Let an automaton $\mathcal{A}$ for σ reach state $\overrightarrow{Q}$ and accumulator values $\langle \overrightarrow{C}, \overrightarrow{D}, \overrightarrow{R} \rangle$ on a prefix of a word w, as well as state $\overleftarrow{Q}$ and accumulator values $\langle \overleftarrow{C}, \overleftarrow{D}, \overleftarrow{R} \rangle$ on the mirror of the corresponding suffix of w. The value N returned by $\mathcal{A}$ on the entire word w is constrained by $N = \phi_g(\overrightarrow{R}, \overleftarrow{R}, \Gamma)$, where Γ is called the *glue expression* and is defined as follows:

1. if $\mathcal{L}_{\overrightarrow{Q}} \mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$ extends σ, then:
   (a) if $\mathcal{L}_{\overrightarrow{Q}} \mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$ extends both the last σ in $\mathcal{L}_{\overrightarrow{Q}}$ and the first σ in $\mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$, then
      $\Gamma = \phi_f(\overrightarrow{C}, \overleftarrow{C}, \overrightarrow{D}, \overleftarrow{D}, \delta_f^i)$;
   (b) if $\mathcal{L}_{\overrightarrow{Q}} \mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$ extends only the last σ in $\mathcal{L}_{\overrightarrow{Q}}$, then $\Gamma = \phi_f(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{D}, \delta_f^i)$;
   (c) if $\mathcal{L}_{\overrightarrow{Q}} \mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$ extends only the first σ in $\mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$, then $\Gamma = \phi_f(\overleftarrow{C}, \overrightarrow{D}, \overleftarrow{D}, \delta_f^i)$;
2. if $\mathcal{L}_{\overrightarrow{Q}} \mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$ creates σ, then $\Gamma = \phi_f(\overrightarrow{D}, \overleftarrow{D}, \delta_f^i)$;
3. if $\mathcal{L}_{\overrightarrow{Q}} \mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$ neither creates nor extends σ, then $\Gamma = \phi_g(\overrightarrow{C}, \overleftarrow{C})$.

Note that these rules are exhaustive and mutually exclusive, because the final conditions of extension and creation are negations of each other.

Example 6. Consider the regular expression $\texttt{Peak} = \text{'}<(<|=)^*(>|=)^*>\text{'}$, the automaton $\mathcal{A}$ in Fig. 2, and the languages in Example 3 for the states of $\mathcal{A}$.

– Consider $\overrightarrow{Q} = m$ and $\overleftarrow{Q} = \ell$: by Example 4, for $L_1 = \mathcal{L}_m$ and $L_2 = \mathcal{L}_\ell^{\text{mir}}$, we know that $\mathcal{L}_{\overrightarrow{Q}} \mathcal{L}_{\overleftarrow{Q}}^{\text{mir}}$ extends only the last $\texttt{Peak}$ in $\mathcal{L}_m$, so rule 1b applies.
– Consider $\overrightarrow{Q} = \ell$ and $\overleftarrow{Q} = \ell$: by Example 5, for $L_3 = \mathcal{L}_\ell$ and $L_4 = \mathcal{L}_\ell^{\text{mir}}$, we know that $\mathcal{L}_\ell \mathcal{L}_\ell^{\text{mir}}$ creates $\texttt{Peak}$, so rule 2 applies.

Table 2. Glue expressions for any $g\_f\_$PEAK constraint. A row index refers to the state of the automaton $\mathcal{A}$ in Fig. 2 reached for the prefix, and a column index refers to the state of $\mathcal{A}$ reached for the mirror of the corresponding suffix.

	k	ℓ	m
k	$\phi_g(\overrightarrow{C}, \overleftarrow{C})$	$\phi_g(\overrightarrow{C}, \overleftarrow{C})$	$\phi_g(\overrightarrow{C}, \overleftarrow{C})$
ℓ	$\phi_g(\overrightarrow{C}, \overleftarrow{C})$	$\phi_f(\overrightarrow{D}, \overleftarrow{D}, \delta_f^i)$	$\phi_f(\overleftarrow{C}, \overrightarrow{D}, \overleftarrow{D}, \delta_f^i)$
m	$\phi_g(\overrightarrow{C}, \overleftarrow{C})$	$\phi_f(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{D}, \delta_f^i)$	$\phi_g(\overrightarrow{C}, \overleftarrow{C})$

– Consider $\overrightarrow{Q} = m$ and $\overleftarrow{Q} = m$: we have that $\mathcal{L}_m = \Sigma^* \mathcal{L}(\texttt{Peak}) \mathcal{L}(=^*)$ and $\mathcal{L}_m^{\mathrm{mir}} = \mathcal{L}(=^*) \mathcal{L}(\texttt{Peak}) \Sigma^*$; note that there does not exist a non-empty suffix of any word in $\mathcal{L}_m$ that, concatenated with a non-empty prefix of any word in $\mathcal{L}_m^{\mathrm{mir}}$, can form a word in $\mathcal{L}(\texttt{Peak})$, so rule 3 applies.

The other six pairs $\langle \overrightarrow{Q}, \overleftarrow{Q} \rangle$ of states are handled similarly. All nine glue expressions are presented in matrix form in Table 2. □

We derived glue constraints for the covered 19 regular expressions: they can be shown to be correct. We establish their propagation impact in Sect. 5.

In the next section, in order to exploit glue constraints better, we provide bounds on their main variables, namely the results of aggregating feature values on a time series, on a prefix thereof, and on the corresponding suffix thereof.

4   Bounds for Time-Series Constraints

We derive bounds on N for any time-series constraint $g\_f\_\sigma(\langle X_1, \ldots, X_n \rangle, N)$ from a few general formulae and the structure of *ground* time series that give extreme values of N. The bounds are valid regardless of the domain choice, but their sharpness is guaranteed only if all the X_i are over the *same* interval domain $[a, b]$. A bound is *sharp* if it equals N for at least one ground time series.

For each regular expression, there exists a necessary condition, based on the domains and number of the X_i, for it to occur at least once within the signature.

Example 7. An `Inflexion`-pattern, called an *inflexion*, within a time series $X = \langle X_1, \ldots, X_n \rangle$ corresponds, except for its first and last elements, to a maximal occurrence of the regular expression `Inflexion` = '$(<(<|=)^*>)|(>(>|=)^*<)$' in the signature of X. The necessary condition for having at least one inflexion in X is $b > a \wedge n \geq 3$, where $[a, b]$ is the smallest interval containing the union of the domains of the X_i. Figure 3a gives an example of inflexion. □

In Sect. 4.1, we describe a systematic methodology for deriving sharp bounds on N for any time-series constraint $g\_f\_\sigma(\langle X_1, \ldots, X_n \rangle, N)$, under the assumptions that all the X_i have the same interval domain and, without loss of generality, that the underlying necessary condition holds. In Sect. 4.2, we illustrate the methodology on one family of constraints.
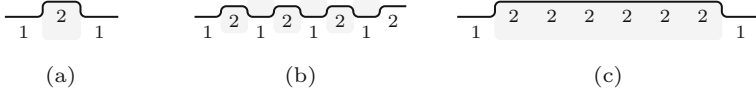
Fig. 3. (a): A time series with an inflexion of shortest width, namely one. (b): A time series with six inflexions. (c): A time series with one inflexion.

4.1   Methodology

For any time-series constraint $g\_f\_\sigma(\langle X_1, \ldots, X_n \rangle, N)$, our aim is to derive formulae for lower and upper bounds on N, parametrised only by n and the domain bounds of the X_i. We define a time-series structure that depends only on g and f, in order to build an *optimal time series* for the upper (resp. lower) bound, defined as a ground time series where N is equal to that upper (resp. lower) bound. We use the following non-mutually-exclusive properties, which were derived manually, all occurrences of 'maximal' and 'minimal' being over all time series of length n over $[a, b]$:

- Property I holds if the number of σ-patterns is maximal.
- Property $\mathrm{II}^{\mathrm{up}}$(resp. $\mathrm{II}^{\mathrm{low}}$) holds if there is at least one σ-pattern whose length is maximal (resp. minimal).
- Property $\mathrm{III}^{\mathrm{up}}_{\max}$ (resp. $\mathrm{III}^{\mathrm{low}}_{\max}$) holds if there is at least one σ-pattern and the absolute difference between b (resp. a) and its maximum is minimal.
- Property $\mathrm{III}^{\mathrm{up}}_{\min}$ (resp. $\mathrm{III}^{\mathrm{low}}_{\min}$) holds if there is at least one σ-pattern and the absolute difference between b (resp. a) and its minimum is minimal.
- Property IV holds if there is no σ-pattern.
- Property $\mathrm{V}^{\mathrm{up}}_{\max}$ (resp. $\mathrm{V}^{\mathrm{low}}_{\max}$) holds if the time series is among those where the sum of the absolute differences between b (resp. a) and the maxima of the σ-patterns is minimal, and the number of σ-patterns is maximal.
- Property $\mathrm{V}^{\mathrm{up}}_{\min}$ (resp. $\mathrm{V}^{\mathrm{low}}_{\min}$) holds if the time series is among those where the sum of the absolute differences between b (resp. a) and the minima of the σ-patterns is minimal, and furthermore the number of σ-patterns is maximal.
- Property $\mathrm{VI}^{\mathrm{up}}_{\max}$ (resp. $\mathrm{VI}^{\mathrm{low}}_{\max}$) holds if the time series is among those where the number of σ-patterns is maximal, and the sum of the absolute differences between b (resp. a) and the maxima of the σ-patterns is minimal.
- Property $\mathrm{VI}^{\mathrm{up}}_{\min}$ (resp. $\mathrm{VI}^{\mathrm{low}}_{\min}$) holds if the time series is among those where the number of σ-patterns is maximal, and furthermore the sum of the absolute differences between b (resp. a) and the minima of the σ-patterns is minimal.
- Property $\mathrm{VII}^{\mathrm{up}}$ (resp. $\mathrm{VII}^{\mathrm{low}}$) holds if there is at least one σ-pattern of maximal length among those with only non-negative (resp. non-positive) elements and the sum of the absolute differences between b (resp. a) and all elements of such a σ-pattern is minimal.
- Property $\mathrm{VIII}^{\mathrm{up}}$ (resp. $\mathrm{VIII}^{\mathrm{low}}$) holds if there is at least one σ-pattern of minimal length and the sum of the absolute differences between b (resp. a) and all elements of such a σ-pattern is minimal.

Table 3. Properties of optimal time series, for feature f and aggregator g.

bound	$g \backslash f$	max	min	one	surface	width
upper	`Max`	$\text{III}^{\text{up}}_{\text{max}}$	$\text{III}^{\text{up}}_{\text{min}}$		$\text{VII}^{\text{up}};\text{VIII}^{\text{up}}$	II^{up}
lower	`Min`	$\text{III}^{\text{low}}_{\text{max}}$	$\text{III}^{\text{low}}_{\text{min}}$		$\text{VII}^{\text{low}};\text{VIII}^{\text{low}}$	II^{low}
upper	`Sum`	$\text{IV};\text{V}^{\text{up}}_{\text{max}};\text{VI}^{\text{up}}_{\text{max}}$	$\text{IV};\text{V}^{\text{up}}_{\text{min}};\text{VI}^{\text{up}}_{\text{min}}$	I	$\text{IV};\text{VII}^{\text{up}};\text{VIII}^{\text{up}}$	$\text{I};\text{II}^{\text{up}}$
lower	`Sum`	$\text{IV};\text{V}^{\text{low}}_{\text{max}};\text{VI}^{\text{low}}_{\text{max}}$	$\text{IV};\text{V}^{\text{low}}_{\text{min}};\text{VI}^{\text{low}}_{\text{min}}$		$\text{IV};\text{VII}^{\text{low}};\text{VIII}^{\text{low}}$	

Twelve constraints have a more involved optimal time-series structure that is not described in this paper for space reasons. The formulae for these twelve constraints take time linear in n to evaluate, whereas the formulae for the constraints covered by the given methodology take constant time to evaluate.

Table 3 gives for each feature/aggregator pair the set of properties of optimal time series. An *optimal time series for a property P* is a ground time series for $g\_f\_\sigma(\langle X_1, \ldots, X_n \rangle, N)$ where N takes the largest (resp. smallest) value for all ground time series possessing P. If there are several properties for an $\langle f, g \rangle$ pair, then we first need to identify an optimal time series for each of those properties. An optimal time series for some property is an optimal time series if it has the maximal (resp. minimal) value of N among the set of optimal time series for every property for $\langle f, g \rangle$.

Example 8. Consider $n = 8$ time-series variables over the integer interval $[1, 2]$.
- Consider SUM_ONE_INFLEXION$(\langle X_1, \ldots, X_8 \rangle, N)$, which constrains N to be the number of inflexions in $\langle X_1, \ldots, X_8 \rangle$. For an upper bound on N, the time series in Fig. 3b is optimal, with $N = 6$ inflexions, and has Property I.
- Consider MAX_MIN_INFLEXION$(\langle X_1, \ldots, X_8 \rangle, N)$, which constrains N to be the maximum of the minima of all inflexions in $\langle X_1, \ldots, X_8 \rangle$. For an upper bound on N, the time series in Figs. 3b and c are optimal, both with $N = 2$, and have Property $\text{III}^{\text{up}}_{\text{min}}$ as both have inflexions whose minima have an absolute difference with $b = 2$ that is 0, hence their minima are b.
- Consider MAX_SURFACE_INFLEXION$(\langle X_1, \ldots, X_8 \rangle, N)$, which constrains N to be the maximum of the sums of the elements of all inflexions in $\langle X_1, \ldots, X_8 \rangle$. By Table 3, for an upper bound on N, there exists an optimal time series for Property VII^{up} or Property VIII^{up} or both. The time series in Fig. 3c is optimal for Property VII^{up}, with $N = 12$: there is an inflexion of maximal length, namely 6, among those with only non-negative elements, and all elements of this inflexion have an absolute difference with $b = 2$ that is minimal, namely 0. The time series in Fig. 3b is optimal for Property VIII^{up}, with $N = 2$: there is an inflexion of minimal length, namely 1, whose elements all have an absolute difference with $b = 2$ that is minimal, namely 0. Hence the upper bound is the maximum of these two values of N, that is 12.   □

4.2   Bounds for Constraints that Only Have Property $\text{III}^{\text{up}}_{\text{min}}$

We consider constraints that only have Property $\text{III}^{\text{up}}_{\text{min}}$, that is with $g = \texttt{Max}$ and $f = \texttt{min}$ according to Table 3. This pair of feature/aggregator makes sense for 18 of the 20 regular expressions in [5]. Our goal is to derive an upper bound on the maximum of the minima of all σ-patterns in a time series, where σ is any of those regular expressions. According to Property $\text{III}^{\text{up}}_{\text{min}}$, in an optimal time series, there is at least one σ-pattern whose minimum is maximal: we use such a σ-pattern to derive this upper bound. For brevity, we do not derive a lower bound, because it is almost always possible to have no σ-patterns at all and the lower bound is then equal to the identity value of g, namely $-\infty$ by Table 1.

Example 9. We can explain the key ideas using Fig. 3b. Consider $\texttt{Inflexion} = $ '$(<(<|=)^{*}>)|(>(>|=)^{*}<)$' and time series over an integer interval $[a, b]$. Our goal is to maximise the maximum of the minima of all inflexions in the time series: in other words, the difference between b and the minimum of some inflexion should be minimal. The time series $t = \langle 1, 2, 1, 2, 1, 2, 1, 2 \rangle$ in Fig. 3b contains two types of inflexions: the first (resp. second) type corresponds to the signature '$<>$' (resp. '$><$'); the inflexions are highlighted in grey. Assume the domain is $[-1, +2]$: the minima of the three '$<>$'-type inflexions equal the domain upper bound, namely $b = 2$, hence the difference with b is 0; the minima of the three '$><$'-type inflexions equal 1, that is $b-1$, hence the difference with b is 1. Hence the smallest difference between b and the minima of the inflexions of t equals 0. Regardless of the value of b, we can always construct a time series with some inflexion that contains b, provided the necessary condition of Example 7 holds. If we now consider the domain $[-1, +5]$, then every element of t can be increased by three, giving $t' = \langle 4, 5, 4, 5, 4, 5, 4, 5 \rangle$, which has the *same* signature as t. As for t, the minima of all '$<>$'-type inflexions equal the domain upper bound, namely $b = 5$, and the minima of all '$><$'-type inflexions equal 4, that is $b-1$. Hence the smallest difference between b and the minima of the inflexions of t' also equals 0. We have shown that the smallest difference between b and the minimum of every inflexion does not depend on b, due to the signature being ground. We need to compute the minimum, denoted by $\Delta_{\texttt{Inflexion}}$, of these smallest differences for *any* signature in $\mathcal{L}(\texttt{Inflexion})$. The sharp upper bound on N for the constraint MAX_MIN_INFLEXION$(\langle X_1, \ldots, X_n \rangle, N)$ equals $b - \Delta_{\texttt{Inflexion}}$.   □

We now formalise these ideas.

Computing the Bounds. Consider a MAX_MIN_$\sigma(\langle X_1, \ldots, X_n \rangle, N)$ time-series constraint where all the X_i are over the *same* interval domain $[a, b]$. Without loss of generality, for determining an upper bound on N, it suffices to restrict our focus on time series containing just *one* σ-pattern, because the result of a Max-aggregation is *any* of its occurrences of the largest value, whereas smaller values are absorbed. Let T_ω denote the set of ground time series over $[a, b]$ whose signature is $\omega \in \mathcal{L}(\sigma)$. For any t in T_ω, let $t_{\downarrow\omega}$ denote the index set of the σ-pattern in t. We want to derive a formula that can be used to evaluate in *constant*

time the upper bound $u = \max_{\omega \in \mathcal{L}(\sigma)} \max_{t \in T_\omega} \min_{i \in t_{\downarrow\omega}} t_i$, which is equal to the wanted upper bound on N under the stated focus restriction. Since u depends also on a and b, its direct computation would not take constant time, because every $|T_\omega|$ depends on a and b. In order to compute u in constant time, we reformulate it as an arithmetic expression on b and a parameter that *only* depends on σ, using the following transformations:

$$u = b - (b - u) = b - (b - \max_{\omega \in \mathcal{L}(\sigma)} \max_{t \in T_\omega} \min_{i \in t_{\downarrow\omega}} t_i)$$

$$= b - \min_{\omega \in \mathcal{L}(\sigma)} (b - \max_{t \in T_\omega} \min_{i \in t_{\downarrow\omega}} t_i)$$

$$= b - \min_{\omega \in \mathcal{L}(\sigma)} \min_{t \in T_\omega} (b - \min_{i \in t_{\downarrow\omega}} t_i) \qquad (1)$$

The value of $\Delta_\omega = \min_{t \in T_\omega}(b - \min_{i \in t_{\downarrow\omega}} t_i)$, called the *shift of signature ω*, does not depend on a and b: every time series t in T_ω that gives this minimum *must* contain b, which can thus be replaced by $\max t$; otherwise, every element of t could be incremented by at least 1, as shown in Example 9, thus reducing the minimal value of $b - \min_{i \in t_{\downarrow\omega}} t_i$ and contradicting the optimal choice of t. Hence $\Delta_\sigma = \min_{\omega \in \mathcal{L}(\sigma)} \Delta_\omega$, called the *shift of regular expression σ*, does not depend on a and b either. The upper bound u on N then is $b - \Delta_\sigma$ by (1). In order to compute Δ_σ, we need to compute Δ_ω for each signature $\omega \in \mathcal{L}(\sigma)$.

We compute each Δ_ω as follows, for a ground signature $\omega = \langle S_1, \ldots, S_\ell \rangle$ linked to a time series $X = \langle X_1, \ldots, X_{\ell+1} \rangle$ by signature constraints. First, we rewrite Δ_ω as follows:

$$\Delta_\omega = \min_{t \in T_\omega}(b - \min_{i \in t_{\downarrow\omega}} t_i) = \min_{t \in T_\omega} \max_{i \in t_{\downarrow\omega}}(b - t_i) \qquad (2)$$

Let Δ_i^t denote $b - t_i$. Note that $\Delta_i^t \geq 0$ because we assume $t_i \leq b$. Hence a time series that minimises the sum of the Δ_i^t also minimises each Δ_i^t, and thus the maximum of the Δ_i^t. So a tuple $\langle \Delta_1^t, \ldots, \Delta_{\ell+1}^t \rangle$ that is minimal for the sum $\sum_{i \in [1,\ell+1]} \Delta_i^t$ is also minimal for $\max_{i \in t_{\downarrow\omega}} \Delta_i^t$ and we can solve the following minimisation problem:

$$\text{minimise} \qquad \sum_{i=1}^{\ell+1} \Delta_i$$

$$\text{subject to} \qquad \Delta_i \geq 0 \qquad \forall i \in [1, \ell+1] \qquad (3)$$

$$\text{if } S_i = \text{`<' then} \quad \Delta_i > \Delta_{i+1} \quad \forall i \in [1, \ell] \qquad (4)$$

$$\text{if } S_i = \text{`=' then} \quad \Delta_i = \Delta_{i+1} \quad \forall i \in [1, \ell] \qquad (5)$$

$$\text{if } S_i = \text{`>' then} \quad \Delta_i < \Delta_{i+1} \quad \forall i \in [1, \ell] \qquad (6)$$

$$\Delta_i \in \mathbb{Z} \qquad \forall i \in [1, \ell+1]$$

The semantics of variable Δ_i, called the *shift of variable X_i*, with $i \in [1, \ell+1]$, is $b - X_i$. For example, if $S_i = \text{`>'}$, meaning $X_i > X_{i+1}$, then $b - X_i < b - X_{i+1}$, hence $\Delta_i < \Delta_{i+1}$. Depending on the value of each S_i, which is assumed ground, we post only one of the constraints (4), (5), or (6) for each pair $\langle \Delta_i, \Delta_{i+1} \rangle$.

Note that Δ_i^t corresponds to $\Delta_i = b - X_i$ when $X_i = t_i$: hence constraint (3). Therefore, in an optimal solution $\Delta^* = \langle \Delta_1^*, \ldots, \Delta_{\ell+1}^* \rangle$, the value of Δ_i^* is the minimal shift of X_i. Hence Δ^* is also an optimal solution to the right-hand side of (2), and so we have $\Delta_\omega = \max_{i \in X_{\downarrow\omega}} \Delta_i^*$. Note that the optimal value of the optimisation problem itself is irrelevant.

Since Δ_ω does not depend on a and b, it can be computed once and for all for any signature ω. Hence it does not matter how much time the minimisation problems take to solve. We show further that the number of minimisation problems and their numbers of variables and constraints can be bounded by very small constants.

Example 10. Consider `Inflexion` $=$ '$(<(<|=)^*>)|(>(>|=)^*<)$' and the signature $\omega = \langle S_1, S_2 \rangle = \langle$'$<$', '$>$'$\rangle \in \mathcal{L}(\texttt{Inflexion})$, linked to the time series $\langle X_1, X_2, X_3 \rangle$. We solve the following minimisation problem to compute Δ_ω:

$$
\begin{aligned}
\text{minimise} \quad & \Delta_1 + \Delta_2 + \Delta_3 \\
\text{subject to} \quad & \Delta_i \geq 0 && \forall i \in [1,3] \\
& \Delta_1 > \Delta_2 \\
& \Delta_2 < \Delta_3 \\
& \Delta_i \in \mathbb{Z} && \forall i \in [1,3]
\end{aligned}
$$

The unique optimal solution is $\langle \Delta_1^*, \Delta_2^*, \Delta_3^* \rangle = \langle 1, 0, 1 \rangle$. The inflexion that corresponds to $\langle S_1, S_2 \rangle$ is $\langle X_2 \rangle$, as exemplified in Fig. 3a, thus $\Delta_\omega = \max_{i \in \{2\}} \Delta_i^* = \Delta_2^* = 0$: this inflexion contains a single element, which can be made to coincide with the domain upper bound. Figure 3a gives an example of such an inflexion within a time series of three variables with 2 as domain upper bound. □

We now state a condition when the computed upper bound is sharp.

Theorem 1. *Consider a time-series constraint* MAX_MIN_$\sigma(\langle X_1, \ldots, X_n \rangle, N)$ *where all the X_i are over the same integer interval $[a,b]$. If at least one word ω in $\mathcal{L}(\sigma)$ with $\Delta_\sigma = \Delta_\omega$ may occur in the signature of $\langle X_1, \ldots, X_n \rangle$, then the upper bound $b - \Delta_\sigma$ on N is sharp.*

Proof. Suppose there exists a word ω that satisfies the stated assumption. Hence there exists a ground time series with an occurrence of ω in its signature: the value of N on such a time series equals $b - \Delta_\omega$, so the bound $b - \Delta_\sigma$ on N is sharp because $\Delta_\sigma = \Delta_\omega$. □

For any regular expression σ in [5] and any time series X over some interval, the assumption of Theorem 1 holds if the necessary condition (such as in Example 7) for having at least one occurrence of σ in the signature of X is met.

Accelerating the Computation of the Shift of a Regular Expression. For some regular expressions, we do not need to minimise over the entire language $\mathcal{L}(\sigma)$ when computing $\Delta_\sigma = \min_{\omega \in \mathcal{L}(\sigma)} \Delta_\omega$. Consider the case when there

exists a word ω in $\mathcal{L}(\sigma)^{\min}$, which is the set of the shortest words of $\mathcal{L}(\sigma)$, such that the following equality holds:

$$\Delta_\sigma = \Delta_\omega \qquad (7)$$

We can then replace $\mathcal{L}(\sigma)$ with $\mathcal{L}(\sigma)^{\min}$ in the definition of Δ_σ. This is the case for all σ in [5], and, additionally, we have $\left|\mathcal{L}(\sigma)^{\min}\right| \leq 2$. Hence computing Δ_σ requires solving at most two optimisation problems over at most four variables.

Example 11. Since `Inflexion` = '$(<(<|=)^*>)|(>(>|=)^*<)$' contains one disjunction at the highest level, every word in $\mathcal{L}(\mathtt{Inflexion})$ belongs to either $L_1 = \mathcal{L}(`<(<|=)^*>`)$ or $L_2 = \mathcal{L}(`>(>|=)^*<`)$. Hence $\mathcal{L}(\mathtt{Inflexion})^{\min}$ is the union of the two sets $L_1^{\min} = \{`<>`\}$ and $L_2^{\min} = \{`><`\}$. Consider the word '$<<>$' in L_1 obtained from the word '$<>$' in $L_1^{\min}$ by inserting just one '$<$'. In order to obtain the minimisation problem for computing $\Delta_{<<>}$, we modify the one of Example 10 for $\Delta_{<>} = 0$ by introducing the new variable Δ_4 and replacing the comparison constraints by the following ones:

$$\Delta_1 > \Delta_2 \;\wedge\; \Delta_2 > \Delta_3 \;\wedge\; \Delta_3 < \Delta_4$$

The unique optimal solution is $\langle 2,1,0,1 \rangle$, giving $\Delta_{<<>} = 1 > \Delta_{<>}$. Similarly, for the word '$<=>$' obtained from '$<>$' by inserting just one '$=$', we have $\Delta_{<=>} = \Delta_{<>}$. Using these base cases, one can prove by induction that the shift of any word in L_1 longer than '$<>$' is at least $\Delta_{<>}$. Applying the same reasoning for the language L_2, we obtain $\Delta_\omega \geq \Delta_{><} = 1$ for all words ω in L_2. Hence $\Delta_{\mathtt{Inflexion}} = \min(\Delta_{<>}, \Delta_{><}) = \min(0,1) = 0$ and equality (7) holds, so we can replace $\mathcal{L}(\mathtt{Inflexion})$ by $\mathcal{L}(\mathtt{Inflexion})^{\min}$ in the definition of Δ_σ. $\qquad\square$

5   Evaluation

We evaluate the impact of the methods introduced in the previous sections on both execution time and the number of backtracks (failures) for all the 200 time-series constraints for which the glue constraint exists.

In our first experiment, we consider a single $g\_f\_\sigma(\langle X_1, X_2, \ldots, X_n \rangle, N)$ constraint for which we first enumerate N and then either find solutions by assigning the X_i or prove infeasibility of the chosen N. For each constraint, we compare four variants of *Automaton*, which just states the constraint, using the automaton of [3]: *Glue* adds to *Automaton* the glue constraints of Sect. 3 for all prefixes and corresponding reversed suffixes, which can be done [6] by just posing *one* additional constraint, namely $g\_f\_\sigma^{\mathrm{mir}}(\langle X_n, \ldots, X_2, X_1 \rangle, N)$; *Bounds* adds to *Automaton* the bound restrictions of Sect. 4; *Bounds+Glue* uses both the glue constraints and the bounds; and *Combined* adds to *Bounds+Glue* the bounds for each prefix and corresponding reversed suffix.

In Fig. 4, we show results for two problems that are small enough to perform all computations for *Automaton* and all variants within a reasonable time. In the first problem (first row of plots), we use time series of length 10 over the
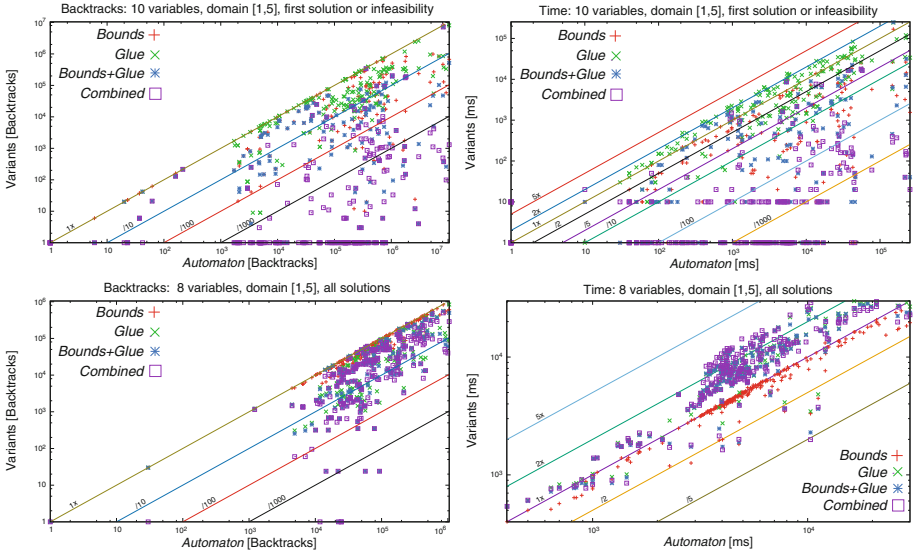
Fig. 4. Comparing backtrack count and runtime for *Automaton* and its variants for the first solution (length 10) and all solutions (length 8).

domain $[1, 5]$, and find, for each value of N, the first solution or prove infeasibility. This would be typical for satisfaction or optimisation problems, where one has to detect infeasibility quickly. Our static search routine enumerates the time-series variables X_i from left to right, starting with the smallest value in the domain. In the case of the initial domains being of the same size, this heuristic typically works best. In the second problem (second row of plots), we consider time series of length 8 over the domain $[1, 5]$, and find all solutions for each value of N. This allows us to verify that no solutions are incorrectly eliminated by any of the variants, and provides a worst-case scenario exploring the complete search tree. Results for the backtrack count are on the left, results for the execution time on the right. We use log scales on both axes, replacing a zero value by one in order to allow plotting. All experiments were run with SICStus Prolog 4.2.3 on a 2011 MacBook Pro 2.2 GHz quadcore Intel Core i7-950 machine with 6 MB cache and 16 GB memory using a single core.

We see that *Bounds* and *Glue* on their own bring good reductions of the search space, but their combinations *Bounds+Glue* and *Combined* in many cases reduce the number of backtracks by more than three orders of magnitude. Indeed, for many constraints, finding the first solution requires no backtracks. On the other hand, there are a few constraints for which the number of backtracks is not reduced significantly. These are constraints for which values of N in the middle of the domain are infeasible, but this is not detected by any of our variants.

The time for finding the first solution or proving infeasibility is also significantly reduced by the combinations *Bounds+Glue* and *Combined*, even though

the glue constraints require two time-series constraints. When finding all solutions, this overhead shows in the total time taken for the three variants using the glue constraints. The bounds on their own reduce the time for many constraints, but rarely by more than a factor of ten.

In our second experiment, shown in Fig. 5, we want to see whether the *Combined* variant is scalable. For this, we increase the length of the time series from 10 to 120 over the domain $[1, 5]$. We enumerate all possible values of N and find a first solution or prove infeasibility. For each time-series constraint and value of N, we impose a timeout of $20\,\mathrm{s}$, and we do not consider the constraint if there is a timeout on some value of N. We plot the percentage of all constraints for which the average runtime is less than or equal to the value on the horizontal axis. For small time values, there are some quantisation effects due to the SICStus time resolution of $10\,\mathrm{ms}$.
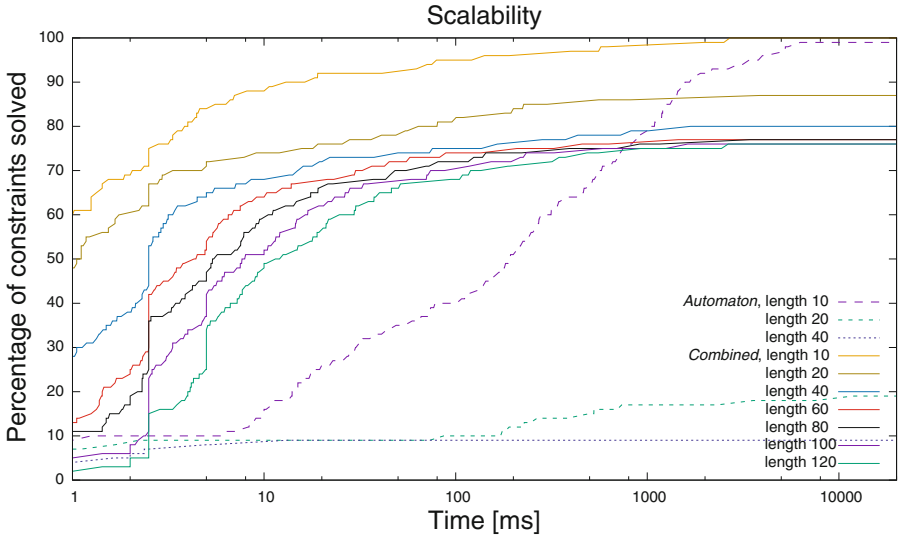
Fig. 5. Scalability results comparing time for *Automaton* and *Combined* on problems of increasing length.

For length 10, we find solutions for all values of N within the timeout, and our plots for *Automaton* (dashed) and *Combined* (solid) reach 100 %, but the average time of *Combined* is much smaller. For *Automaton*, the percentage of constraints that are solved within the timeout drops to less than 20 % for length 20, and less than 10 % for length 40. For *Combined*, we solve over 75 % of all constraints within the time limit, even for lengths 100 and 120.

The constraints that are not solved by *Combined* use the feature `surface` or the aggregator `Sum`. The worst performance is observed for constraints combining both `surface` and `Sum`. This is not surprising, as we know that achieving domain consistency for many of those constraints is NP-hard (encoding of *subset-sum*).

6   Conclusion

For the time-series constraints in [5], specified by a triple $\langle \sigma, f, g \rangle$, we showed in [3] how to generate simplified automata and linear implied constraints. Here, we further enhance the propagation of time-series constraints by a systematic generation of bounds and glue constraints. Rather than finding bounds and glue constraints for each time-series constraint independently, we introduce the concepts of *parametric bounds* and *parametric glue constraints*. Our approach differs from existing ones, which design dedicated propagation algorithms [4,14] and reformulations [9,10] for specific constraints, or propose generic approaches [13,15] that do not focus on the combinatorial aspect of a constraint.

References

1. Almeida, M., Moreira, N., Reis, R.: Enumeration and generation with a string automata representation. Theor. Comput. Sci. **387**(2), 93–102 (2007). The FAdo tool is available at http://fado.dcc.fc.up.pt/
2. Arafailova, E., Beldiceanu, N., Carlsson, M., Douence, R., Flener, P., Francisco Rodríguez, M.A., Pearson, J., Simonis, H.: Global constraint catalog, volume ii: time-series constraints. Technical report, Computing Research Repository (forthcoming). http://arxiv.org
3. Arafailova, E., Beldiceanu, N., Douence, R., Flener, P., Francisco Rodríguez, M.A., Pearson, J., Simonis, H.: Time-series constraints: improvements and application in CP and MIP contexts. In: Quimper, C.-G., Cavallo, M. (eds.) CPAIOR 2016. LNCS, vol. 9676, pp. 18–34. Springer, Heidelberg (2016). doi:10.1007/978-3-319-33954-2_2
4. Beldiceanu, N., Carlsson, M.: Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 377–391. Springer, Heidelberg (2001)
5. Beldiceanu, N., Carlsson, M., Douence, R., Simonis, H.: Using finite transducers for describing and synthesising structural time-series constraints. Constraints **21**(1), 22–40 (2016). Journal fast track of CP 2015: summary on p. 723 of LNCS 9255, Springer (2015)
6. Beldiceanu, N., Carlsson, M., Flener, P., Rodríguez, M.A.F., Pearson, J.: Linking prefixes and suffixes for constraints encoded using automata with accumulators. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 142–157. Springer, Heidelberg (2014)
7. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 107–122. Springer, Heidelberg (2004)
8. Beldiceanu, N., Ifrim, G., Lenoir, A., Simonis, H.: Describing and generating solutions for the EDF unit commitment problem with the ModelSeeker. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 733–748. Springer, Heidelberg (2013)
9. Bessiére, C., Hebrard, E., Hnich, B., Kiziltan, Z., Quimper, C.-G., Walsh, T.: Reformulating global constraints: the SLIDE and REGULAR constraints. In: Miguel, I., Ruml, W. (eds.) SARA 2007. LNCS (LNAI), vol. 4612, pp. 80–92. Springer, Heidelberg (2007)

10. Bessiére, C., Katsirelos, G., Narodytska, N., Quimper, C.-G., Walsh, T.: Decomposition of the NVALUE constraint. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 114–128. Springer, Heidelberg (2010)
11. Francisco Rodríguez, M.A., Flener, P., Pearson, J.: Implied constraints for Automaton constraints. In: Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.) GCAI 2015. EasyChair Proceedings in Computing, vol. 36, pp. 113–126 (2015)
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley, Boston (2007)
13. Monette, J.-N., Flener, P., Pearson, J.: Towards solver-independent propagators. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 544–560. Springer, Heidelberg (2012)
14. Régin, J.C.: A filtering algorithm for constraints of difference in CSPs. In: Hayes-Roth, B., Korf, R.E. (eds.) AAAI 1994, pp. 362–367. AAAI Press (1994)
15. Van Hentenryck, P., Saraswat, V., Deville, Y.: Design, implementation, and evaluation of the constraint language cc (FD). Technical report CS-93-02, Brown University, Providence, USA, January 1993. Revised version in Journal of Logic Programming 37(1–3), 293–316 (1998). Based on the unpublished manuscript Constraint Processing in cc (FD) (1991)

An Adaptive Parallel SAT Solver

Gilles Audemard, Jean-Marie Lagniez, Nicolas Szczepanski[(✉)],
and Sébastien Tabary

Univ. Lille-Nord de France, CRIL/CNRS UMR 8188, Lens, France
{gilles.audemard,jean-marie.lagniez,nicolas.szczepanski,
sebastien.tabary}@cril.fr

Abstract. We present and evaluate AMPHAROS, a new parallel SAT solver based on the divide and conquer paradigm. This solver, designed to work on a great number of cores, runs workers on sub-formulas restricted to cubes. In addition to classical clause sharing, it also exchange extra information associated to the cubes. Furthermore, we propose a new criterion to dynamically adapt both the amount of shared clauses and the number of cubes. Experiments show that, in general, AMPHAROS correctly adjusts its strategy to the nature of the problem. Thus, we show that our new parallel approach works well and opens a broad range of possibilities to boost parallel SAT solver performances.

1 Introduction

Papers dealing with SAT solvers usually begin by recalling the tremendous progress achieved on problems coming from industry. Recent results are indeed very impressive, and a large number of industrial problems are nowadays solved using a reduction to SAT instead of ad-hoc solvers [11,35,40]. However, playing the devil's advocate, one can observe that progress has slowed down noticeably. It has become harder and harder to improve solvers dramatically. Furthermore, SAT suffers from its own success, since formulas to solve are more and more difficult.

At the same time, cloud computing is changing the landscape of computing science: it is now possible to request a virtually unlimited number of computing units that can be used within a few seconds. However, as it was pointed out during the last competition [37], parallel SAT solvers are not well scalable. Indeed, the winner of the parallel SAT track chose to only use half of the available cores. Thus, to benefit from the huge number of computing units, as in a cloud context, one must design new solvers architectures.

In the case of SAT solving, solvers can be divided into two categories. First and foremost, portfolio based approaches [1,8,13,23,24,36] run different strategies/heuristics concurrently, each on the whole formula. While computing the processes exchange information (generally in the form of learnt clauses) to help each other [1,7,23,24]. The second category of solvers uses the well known

---

divided and conquer paradigm [2, 15, 16, 25, 26, 39, 41, 42]. In such solvers, the search space is divided into sub-spaces, which are successively sent to SAT solvers running on different processors, so called workers. In general, each time a solver finishes its job (while the others are still working), a load balancing strategy is invoked, which dynamically transfers sub-spaces to this idle worker [15, 16]. The sub-spaces can be defined using the guiding path concept [42], generated statically, *i.e.*, before the search [25, 39], or dynamically, *i.e.*, during the search process [2, 26, 41]. As in portfolio solvers, learnt clauses can also be shared [18].

Even though the winners of the parallel track of the last SAT competitions are based on the portfolio paradigm, solvers based on the divide and conquer approach become increasingly more efficient (TREENGELING [12] a solver based on this paradigm was ranked second in the last competition). It is in this context that we propose AMPHAROS, a new parallel SAT solver, which follows the divide and conquer approach. Our long term objective is to develop a SAT solver for the cloud and this paper is a first step in this direction. In our approach, the formula is partitioned using cubes (as in [41]). One process, named MANAGER, is dedicated to managing these cubes. Then, solvers work on the formulas induced by those cubes. In contrast to other divide and conquer approaches, several solver may work on the same sub-problem and they can stop working before finding a solution or a contradiction. The latter is to avoid solvers being stuck on instances that turn out to be too hard for them. In that case, the solver asks the manager for another sub-problem. This sub-problem can either originate from an existing cube or from refining the current sub-problem. In our approach, the solvers select by themselves the dynamically generated cubes they try to solve. Additionally, two types of learnt clauses are shared: the classical shared clauses and others that are dependent on the cubes.

Since our goal is to solve SAT with a great number of computing units, it is important to propose a parallel architecture which adapts its strategy to the number of workers and the nature of the problem. To this end, we propose an approach which uses an adaptive algorithm that adjusts simultaneously and dynamically the number of clauses that are shared and the number of new cubes. This is possible thanks to a new measure that estimates if the search process has to be intensified or diversified. As we demonstrate in experiments, this measure works well and aligns with the stated goal. We show that when the search space needs to be diversified (resp. intensified), the proposed measure detects that the number of cubes must be increased (resp. decreased) and the number of shared clauses decreased (resp. increased).

2   Preliminaries

Due to lack of space, we assume the reader to be familiar with the essentials of propositional logic and SAT solving. Let us just recall some aspects of CDCL SAT solvers [30, 32]. CDCL solving is a branching search process, where at each step a literal is selected for branching. Usually, the variable is picked w.r.t. the VSIDS heuristic [32] and its value is taken in a vector, called polarity vector,

in which the previous value assigned to the variable is stored [34]. Afterwards, Boolean constraint propagation is performed. When a literal and its opposite are propagated, a conflict is reached, a clause is learnt from this conflict [30] and a backjump is executed. These operations are repeated until a solution is found or the empty clause is derived.

CDCL SAT solvers can be enhanced by considering restart strategies [20] and deletion policies for learnt clauses [3,6,19]. Among the measures proposed to identify the relevant clauses, the literal blocked distance measure (in short LBD) [6] is one of the most efficient. The clause's LBD corresponds to the number of different levels involved in a given learnt clause. Then, as experimentally shown by the authors of [6], clauses with smaller LBD should be considered more relevant.

It is well known that for several applications it is necessary to solve many similar instances [5,9,17]. To make solvers more effective in such a context, it is particularly useful to use assumptions to keep track of learnt clauses during the whole search. A set of assumptions is defined as a set of literals that are assumed to be true [17]. This set can be viewed as a cube, i.e. a conjunction of literals (in the remainder of this paper, we denote cubes using square brackets, also we sometimes identify cubes with the formulas they imply), and the search is restricted to this cube. If during the search process, one needs to flip the assignment of one of these assumptions to false, the problem is unsatisfiable under the initial assumptions. In such a situation, it is possible to recursively traverse the implication graph to extract a clause that explains the reason of the conflict. Even if this problem seems close to the classical SAT problem, a special track of the last SAT competition has been dedicated to this issue [37] and several existing studies attempt to improve SAT solvers to deal with assumptions [4,28,33].

3   Tree Management

The performance of divide and conquer approaches depends on both, the quality of the search space splitting, and how the sub-spaces are assigned to the solvers.

Even if AmPharoS is a divide and conquer based solver, it is important to stress that, contrary to [38], it does not use the work stealing strategy. In our case, the division is done in a classical way as in [2,16]. More precisely, our approach generates guiding paths, restricted to cubes, that cover all the search space. This way, the outcome of the division is a tree where nodes are variables and the left (resp. right) edge corresponds to the assignment of the variable to true (resp. false). Then, solvers operate on leaves (represented by the symbol NIL) and solve (under assumptions) the initial formula restricted to a cube which corresponds to the path from the root to the related leaf. Figure 1a shows an example of a tree containing three open leaves (cubes $[x_1, \neg x_2, x_4]$, $[x_1, \neg x_2, \neg x_4]$ and $[\neg x_1, \neg x_3]$), two closed branches (already proven unsatisfiable) and four solvers ($S_1 \ldots S_4$) working on these leaves.

As we will see in Sect. 3.2, in our architecture, solvers can work on the same cube (as solvers S_1 and S_2 in Fig. 1a) and can stop working before finding a solution or a contradiction. In AmPharoS, each time a solver shares information or

asks to solve a new cube, it communicates with a dedicated worker, called MAN-AGER. Its main mission is to manage the cubes and the communication between the solvers (here CDCL solvers). Thus, when a solver decides to stop solving a given cube (without having solved the instance), it can ask the MANAGER to enlarge this one (see Sect. 3.3). Another situation where a solver stops, is once a branch is proved to be unsatisfiable. In this case, a message informs the MAN-AGER and the tree is updated in consequence (see Sect. 3.4). In both cases, when a solver stops it goes through the tree and starts solving a new cube (potentially the same, see Sect. 3.2). The end of the solving process finally occurs either when a cube is proved to be satisfiable or when the tree is proved to be unsatisfiable.

This section describes the overall picture of our solver. First, in Sect. 3.1, the way the tree is initialized is presented. Then, the transmission and extension processes are respectively explained in Sects. 3.2 and 3.3. Finally a tree pruning rule is introduced in Sect. 3.4.

3.1   Initialization

At the beginning of the search process, we initialize the workers. This step is required to setup the activity (related to VSIDS heuristic [32]), the polarity of variables and to create the root of the tree. To this end, all solvers try to solve the whole formula concurrently until a given amount of conflicts is reached (10,000 in our implementation). Note that this corresponds to solve an empty cube. In order to avoid performing the same search, the first descent of each solver (*i.e.* the choice of the variables and their polarity on the first branch) is randomized. Then, in the same manner as [31], the first solver reaching the maximum number of conflicts communicates its best variable with respect to the VSIDS heuristic to the MANAGER. This variable becomes the root of the tree. Consequently, the tree only contains two leaves, *i.e.* cubes are restricted to a single literal (a variable and its opposite). Regarding Fig. 1a, the selected variable was x_1 and the set of initial cubes was $\{[x_1], [\neg x_1]\}$.

3.2   Transmission

As already mentioned, a solver may stop the search before solving its instance. This situation occurs when it cannot solve the sub-problem associated to the cube with a number of conflicts less than a certain limit (10,000 in our implementation). The solver then contacts the MANAGER in order to select a potentially new cube to solve. The originality of our method is that a solver selects by itself one cube among all unsolved ones in the tree (corresponding to NIL leaves).

Figure 1 shows a diagram sequence (Fig. 1b) that illustrates the exchanged messages when the solver S_4 requests a new cube from the MANAGER's tree (Fig. 1a).

A first message (GO-ROOT) is sent by the solver to ask for the root of the tree. It receives x_1. Then, at each step of the cube selection, the solver asks for the children of the previously received variable (with message GIVE-CHILDREN). The answer is composed of two triplets: one for each polarity of the current

(a) Current tree   (b) Sequence diagram
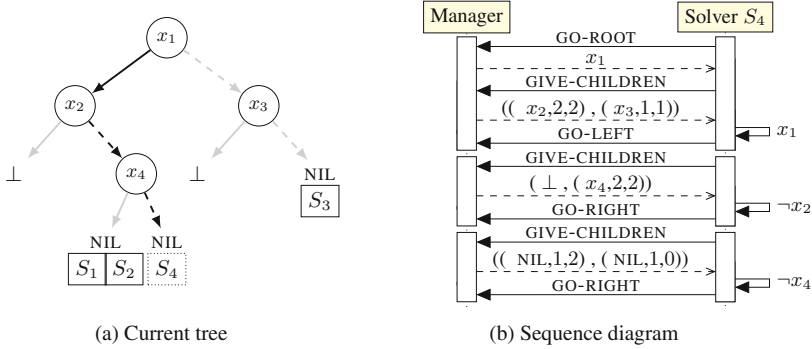
Fig. 1. Schematic overview of how the solver S_4 and the MANAGER interact to select a cube in the current tree represented (a) (a plain (resp. dotted) line means that the variable is assigned to true (resp. false)). On the sequence diagram, in (b), we can see that seven messages are exchanged between the MANAGER and the solver before S_4 starts to solve the sub-problem induced by $[x_1, \neg x_2, \neg x_4]$. The path selected by S_4 is represented with black lines in the left picture.

node. Each triplet is composed of the child variable, the number of available leaves (NIL nodes) and the number of solvers working on these leaves, in that order. Considering Fig. 1b, the first message returns the triplet $(x_2, 2, 2)$ for the positive polarity of x_1 (the left branch contains two leaves and two solvers (S_1 and S_2)) and $(x_3, 1, 1)$ for the negative one.

The solver decides to go down either on the left (assigning positively the current variable) or on the right (assigning negatively the current variable) branch according to the values returned in these triplets. By default, it selects the branch where the number of working solvers is lower than the number of leaves. The idea is to cover the most of cubes and to dispatch solvers all over the tree. If this condition is true or false for both branches, the solver selects the branch according to its polarity vector [34]. Note that in this implementation, we do not know if some cubes do not contain solvers. After selecting its branch, the solver informs the MANAGER (with messages GO-LEFT or GO-RIGHT) and assigns the related literals using assumptions.

Thus, in our example of Fig. 1, the solver S_4 assigns x_1 (the root) positively using its polarity (as the condition previously mentioned is false for both branches). Since the branch related to x_2 is already proven unsatisfiable, S_4 does not have other alternatives to setting the literal x_2 to false. Finally, it has to set x_4 to false since the previous condition holds. Arriving at a leaf, the solver starts to solve the cube $[x_1, \neg x_2, \neg, x_4]$.

3.3   Extension

Initially, the tree contains only one variable and then two cubes to solve (see Sect. 3.1). To divide the original formula into subproblems, we propose to

dynamically extend the tree during the search. Recall that we do not use the work stealing strategy.

One associates to each leaf an integer variable β representing the presumed difficulty of a subproblem (cube). Each time a solver cancels its search on a given cube (associated with a leaf of the tree), the variable β of this leaf is incremented. Then, a large value of β expresses that a cube is potentially hard to solve. Note that a solver can increase several times the same variable β. When a solver stops its search and requests a new cube, the MANAGER increments the value β associated to the leaf on which the solver was working. When the β value of a leaf is greater or equal than the number of open leaves (*i.e.* NIL leaves) times an extension factor f_e then the tree is expanded on the given leaf.[1]

The extension is done in the following way. The last solver increasing the variable β returns its best boolean variable w.r.t. to VSIDS heuristic and two new leaves are created, extending the related cube. The β values of the two leaves are initialized to 0. Taking into account the number of open leaves, the more unsolved cubes the tree contains, the less extensions are performed. In this way and contrary to Cube And Conquer [41], our approach does not create too many cubes, regardless of the number of cubes already proven unsatisfiable. Since a leaf can contain many solvers, note that after extension, some solvers can work on a node that is not a leaf.

(a) Extension requested by the solver S_3 (b) Extension accepted by the MANAGER
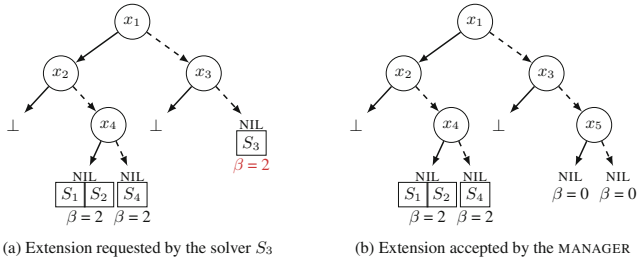
Fig. 2. The left picture represents the tree before the S_3's extension request was accepted. Since the value of β associated to the node satisfied the extension criterion, the MANAGER accepts this extension and modified the left tree to obtain the right one. (Color figure online)

Figure 2 shows an example of an extension. The tree (the same as in Fig. 1) contains 3 open leaves and some solvers work on these leaves. When, solver S_3 stop working on cube $[\neg x_1, \neg x_3]$ the associated β (in red) is incremented and becomes 3. The condition allowing an extension holds (we suppose that f_e is equal to one) and thus extension is performed. Solver S_3 that is responsible for the extension provides its best variable (x_5) to the MANAGER and the cube $[\neg x_1, \neg x_3]$ is expanded with the variable x_5 generating two new cubes. Note that the (red) β value initiator of this extension becomes useless since its associated

---

[1] We will discuss about the definition of the extension factor in Sect. 5.2.

node is not a leaf anymore. Solver S_3 is now free to ask the MANAGER a new cube to solve (see Sect. 3.2). Furthermore, in the next step, no matter which solver ask for extension, it will not be performed since the number of leaves is now equal to 4 (we suppose here that f_e remains unchanged and is still equal to 1).

3.4   Pruning

Because each sub-problem is solved under assumptions, when a cube is proved to be unsatisfiable, the solver (from which unsatisfiability is proved) computes a conflict clause (which is the negation of a subset of the literal assumptions). This information is transfered to the MANAGER which is able to compute a cutoff level in the tree search. The tree is simplified in consequence. Let us remark that a solver can directly prove the global unsatisfiability of the problem when the computed conflict clause is empty.

Moreover, if both children of a node are unsatisfiable then this node also becomes unsatisfiable. In that case, the node can be safely removed and the unsatisfiability is directly associated to the edge of its parent. Of course, this process is recursively applied until each node has at least one non-unsatisfiable child.

4   Clause Exchange

In this section, we discuss the two ways of exchanging information in our solver AMPHAROS. We first explain how the clauses learnt by a solver are shared with the others and then we present an original approach to sharing local unit literals by taking advantage of our tree.

4.1   Classical Clause Sharing

It is well known that clause sharing noticeably improves the performance of parallel SAT solvers [24]. In our framework, solvers also share learnt clauses. However, contrary to the classical behavior where the clauses are directly shared between workers, for us information passes through the MANAGER.

Clause sharing from the solver side. Once a solver reaches a threshold of conflicts (500 in our implementation), it communicates with the MANAGER to send and/or receive a set of clauses. Clauses to be sent are saved in a buffer which is cleared after each communication with the MANAGER. Good clauses with respect to initial LBD (less or equal to 2) are directly added to the buffer. Other clauses are also added, as in [7], if they participate in the conflict analysis. However, because we cannot share as many clauses as SYRUP, only clauses which obtain a dynamic LBD less or equals to 2 before being used twice in the conflict analysis procedure are shared.

In order to deal with imported clauses, solvers manage three buffers: `standby`, `purgatory` and `learnt`. Received clauses are stored in `standby`. In this buffer, clauses are not attached to the solver [3]. Every 4,000 conflicts, clauses are reviewed: they can be transfered from a buffer to another, or be definitively deleted or kept in the current buffer. A clause from the `standby` buffer can be transfered to the `purgatory` buffer. Contrary to the `standby` buffer, clauses in the `purgatory` are attached to the solver and then participate to the unit propagation process. We discuss the criterion allowing a clause to be moved from `standby` to `purgatory`in Sect. 5.2. In the same manner, a clause from the `purgatory` can be transfered to the third buffer `learnt` when it is used at least once in the conflict analysis process. The temporary buffer `purgatory` is used to limit the impact of new clauses on the learnt strategy reduction. The reduction strategy used to clean these two additional buffers depends on a counter associated with each clause. The counter is incremented each time the associated clause remains in the same buffer. If the counter reaches a threshold (14 in our implementation), the clause is deleted. Note that the counter is reset each time a clause is moved from one buffer to another.

Clause sharing from the manager side. MANAGER collects learnt clauses from every solver and manages them. Learnt clauses are stored in a queue and the MANAGER periodically checks if they are subsumed or not. In practice, a single core is dedicated to the MANAGER. Thus, processing all clauses in the queue at once can be time consuming and can block communications between MANAGER and solvers. To avoid this situation, MANAGER checks subsumption by batches of 1,000 clauses each. MANAGER stores the learnt clauses that are not subsumed in a database and sends them each time a solver requests them. Of course, sent clauses are those that have not been already sent to the solver and that are not coming from it.

4.2   Assumptive Unit Literals

A second way of exchanging information in our approach is transferring unit literals (which are propagated under some assumptive literals) between solvers and the MANAGER. In the following, we present where these literals originate from and how they are exchanged and managed.

Assumptive unit literals from the solver side. Let us first recall that each solver works under an assumption A (this assumption can be empty) representing the cube to solve. When a literal $\ell \notin A$ is propagated thanks to a sub-assumption $A' \subseteq A$, this information can be spread to the MANAGER in order to be broadcasted to other solvers. More precisely, the solver communicates to the MANAGER that ℓ can be propagated with A'. From the other side, when a solver selects a branch (i.e. a literal ℓ') during cube transmission, it also receives the set of unit literals associated with ℓ' that can be propagated. Thus, the transmission of a cube (see Sect. 3.2) contains these additional messages. Hence, MANAGER takes care of a decorated tree containing guiding paths and the set of unit literals that

(a) S_4 gives $\neg u_2$ $\neg u_4$ u_5 to MANAGER

(b) Unit literals management
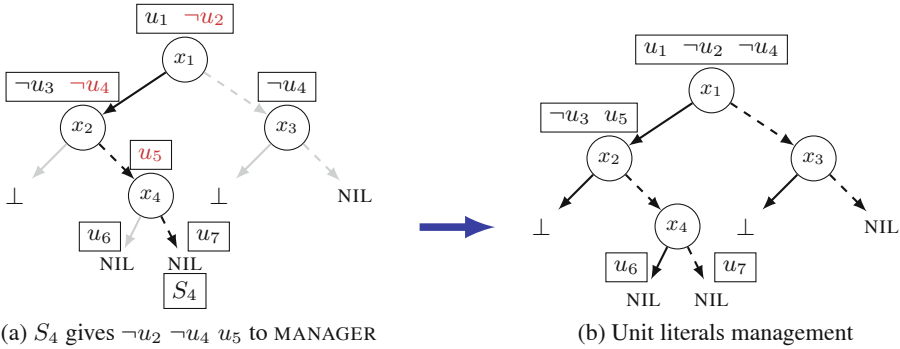
Fig. 3. The left picture represents a decorated tree with the additional literals (in red) given by S_4. These additional literals are pull up, using the unsat (pull up u_5) and identical literals (pull up u_4) rules, to obtain the right tree. (Color figure online)

have to be propagated at each branch. Figure 3a shows an example of such a tree. When solver S_4 asks for a branch, it starts by recovering the set of unit literals $\{u_1\}$. It also propagates $\neg u_2$ (in red) giving this information to the MANAGER. It selects the branch x_1 and then, retrieves the literal $\neg u_3$ to propagate. In the same way, it also propagates $\neg u_4$, providing such assumptive unit literal to the MANAGER and so on.

As we will see in the experiments later, assumptive literals are very important. They are special clauses that clearly reduce the search space of a given branch. Consequently, the fact that a literal ℓ can be propagated from A' is taken into account in the solver by adding, in a dedicated database (this database is different from the aforementioned `learnt` buffer and is never cleaned up), a clause built with the negation of A' and the literal ℓ'. Remark that when $A' = \emptyset$ the literal ℓ' is unit and is added to the unit literals of the solver.

Assumptive unit literals from the manager side. When the MANAGER learns that a literal can be propagated from a subset of literals coming from an assumption, this information is communicated during cube's transmission and can be added in the last branch of the node associated with this sub-assumption. From this decorated tree, one can pull up unit literals from a branch to higher branches. This situation occurs either when a branch is proved unsatisfiable or when both branches of a node contain the same literal [29] (as highlighted Fig. 3). In the first case, all the literals of the non-unsatisfiable branch are pulled up to the father branch (as literal u_5). In the second case, only literals occurring in both branches are transmitted to the father branch (this is the case for literal $\neg u_4$). This process loops recursively until a fix point is reached. Remark that when no father branch exists (occurring when literals are moved from branches of the root node) then these literals are proved unit.

5 The Intensifcation/Diversification Dilemma

When several solvers run concurrently on a problem, they can perform redundant work. Identifying such a situation, it would be beneficial to modify the solvers' strategies in order to diversify the search. Nevertheless, due to clause sharing between solvers, exploring too different search spaces is also a handicap. Thus, in some situation focusing several solvers on the same part is required (intensification).

This paradigm, called intensification/diversification dilemma, has already been studied in the context of portfolio-based parallel SAT solvers. This issue can be addressed either statically, by using several solvers with orthogonal strategies [1, 24, 36], or dynamically, by modifying the solvers' strategies during the search. However, deciding when a solver must intensify or diversify its search is not easy and only few publications tried to deal with this problem [21, 22]. Thus, in [21], a master/slave architecture is proposed, where masters try to solve the original problem (ensuring diversification), whereas slaves intensify their master's strategy. In [22], a measure to estimate the degree of redundancy between two solvers is presented. It considers that two solvers are closed when they have approximately the same polarity vector. The diversification process consists in modifying the way the phase of the next decisions is realized.

To the best of our knowledge, no criterion has been established to identify that several solvers execute redundant work except the measure based on the polarity mentioned before [22]. Unfortunately, this criterion is not applicable with many solvers (this measure has been initially proposed for a portfolio of four solvers). That is why a more scalable criterion is required.

5.1 Evaluating the Degree of Redundancy

We propose to measure the degree of redundancy by taking into account how many clauses that are shared between solvers are redundant. We use a list to store from the beginning the number of received clauses (st_r) and a second to store the number of kept clauses (st_k). Kept clauses are those which have not been removed during the subsumption process. Each time a solver comes back to the MANAGER (every 1,000 conflicts in our implementation), it shares its clauses. The number of received (resp. kept) clauses since the beginning is pushed to st_r (resp. st_k) by the MANAGER.

The *redundancy shared clauses measure*, in short *rscm*, is defined for a step t w.r.t. a sliding window of size m (20,000 in our experiments) as the ratio between the number of clauses received during the last $t - m$ updates of st_r and the number of clauses kept during the same time. More precisely, we have $\forall j < 0,\ st_r[j] = st_k[j] = 0$:

$$
\begin{aligned}
rscm_t &= \frac{st_r[t] - st_r[t-m]}{st_k[t] - st_k[t-m]}, \text{ if } st_k[t] - st_k[t-m] \neq 0 \\
rscm_t &= st_r[t] - st_r[t-m], \text{ otherwise}
\end{aligned}
\tag{1}
$$

First, note that when several solvers work on the same part of the search space, there is a high likelihood that learnt clauses by the different solvers are redundant. This means that the number of subsumed clauses is important and therefore the *rscm* value is high. Conversely, when solvers are sparsed in the search space, there is a high probability that shared clauses are not redundant and then the *rscm* value tends to be low. Consequently, a small value of the *rscm* indicates that the solver needs to intensify the search, whereas a high value signifies that the solvers have to diversify their search space.

5.2   Intensification/Diversification Mechanisms Based on the *rscm* Measure

It is possible to control in several ways how solvers explore the search space (shared clauses, solvers' heuristics, ...). In AMPHAROS, we choose to solve the intensification/diversification dilemma by controlling two criteria: the way the tree is extended (see Sect. 3.3) and the number of clauses which are transferred from the `standby` to the `purgatory` buffers (see Sect. 4.1). Thus, for us, diversifying (resp. intensifying) the search consists in increasing (resp. decreasing) these two parameters. Before introducing them, let us a summarize:

Few subsumed Clauses (*rscm* is low)	Many subsumed clauses (*rscm* is high)
Reduce extension	Favour extension
Increase the number of imported clauses	Limit the number of imported clauses
Intensification	Diversification

Extension guiding by the *rscm*. First, let us remark that each path from the root to a leaf represents a unique set of literals that splits the search space in a deterministic way. Thus, the bigger the tree, the higher the probability to run two solvers in totally different sub-problems. To control the tree grows, we define the extension factor fe introduced in the Sect. 3.3 in the following way:

$$fe_t = \frac{1,000}{rscm_t{}^3} \tag{2}$$

Let us recall that this extension factor is used to define the threshold of misses that a cube can encounter before an extension is accepted. Hence, the smaller (resp. bigger) the $rscm_t$ value is, the bigger (resp. smaller) the fe value is and then the slower (faster) the tree extension is. Note that the cubic factor allows to decrease fe rapidly while fe is bounded (by 1,000) since when fe is high solvers run in concurrently and the tree is never updated. To prevent the tree from growing too quickly, we also bound the minimum value that fe can take by 10.

Condition to move from the `standby` to the `purgatory`. When a clause is received by a solver it is possible that it is subsumed by a clause already present. This becomes highly probable when almost all shared clauses are found to be subsumed during the clause subsumption process. Thus, it seems natural

that the number of accepted clauses (i.e. the number of clauses transferred from the `standby` to the `purgatory`) increases (resp. decreases) when the *rscm* value decreases (resp. increases).

As already mentioned (Sect. 4.1), in AMPHAROS the clauses freshly received are not directly attached to the solver. Thus, it is important to choose a clause selection criterion independant from the activation of clauses. To control the amount of clauses moved from the `standby` to the `purgatory` we use the notion of *psm* introduced in [3] and already used in the portfolio based SAT solver PENELOPE [1]. Recall that the *psm* of a clause represents the number of literals which are assigned to true by the polarity vector. Thus, a clause can be scored even if it is not used by the solver.

Then, in order to increase/decrease the number of clauses attached (and then transferred) in the `purgatory`, a criterion based on both the *psm* and *rscm* values is proposed. This criterion is motivated by the observation from [3] that the clauses with a small *psm* value have a great potential to enter in conflict or be used during the search. Thus, a clause will be authorized to move from the `standby` buffer to the `purgatory` buffer when its *psm* value is less or equals than $\lfloor \frac{psm_{max}}{rscm_t} \rfloor$, where psm_{max} corresponds to the *psm* maximum limit accepted (set to 6 in our experiments). Consequently, clauses with a *psm* value of zero will be systematically accepted whatever the value of *rscm*. Whereas, clauses with a high *psm* value will be accepted if and only if they are probably not subsumed.

6 Experiments

We now evaluate AMPHAROS on the 100 benchmarks from the SAT-RACE 2015, parallel track [37]. During the last competition 53 (resp. 33) instances have been proved satisfiable (resp. unsatisfiable) by at least one solver and 14 instances remained unsolved. All experimentations have been conducted on 2 Dell R910 with 4 Intel Xeon X7550. Each node has 32 cores, a gigabit ethernet controller and 256 GB of RAM. Time limit was set to 1, 200 s per test (wall clock time). Then, for experiments executed with 64 cores, we use two different computers. All log files and additional pictures are available in http://www.cril.univ-artois. fr/ampharos/.

6.1 Communication Management

Since in AMPHAROS a lot of messages have to be exchanged between the MANAGER and solvers, the management of the communications has to be very effective. Thus, we have opted for the open source Message Passing Interface implementation (Open MPI) to manage the communication on a low level. The bottleneck imposed by the fact that the MANAGER has to all at once compute the subsumed clauses and communicate with the solvers, was a major problem. To avoid that solvers wait too long without work, a round robin architecture with non-blocking listening of solvers was put in place. Moreover, because the subsumption process can be time consuming, the clauses received to be checked

are not treated at once (in our implementation packet of 1,000 clauses are considered). Thus, the MANAGER communicates with a solver, then checks a set of clauses, and so on, until the time limit is reached or the problem is proved satisfiable/unsatisfiable.

6.2   Setup

AMPHAROS is a modular framework that allows to add easily new types of solvers. For these experiments three sequential SAT solvers have been used: GLUCOSE [6], MINISAT [17] and MINISATPSM [3]. Only a couple small changes have been implemented in these solvers. In order to manage the interactions with the MANAGER, all solvers implement a C++ interface. This interface grouped communication routines and methods used to deal with `standby` and `purgatory` buffers. The core of solvers has also been modified in order to avoid resetting everything at each call to SAT solver (restart, learnt deletion policies, . . .). Moreover, as for the version of GLUCOSE presented in [4], when a solver restarts it does not go to decision level 0 but to the level of the last assumption. The clauses moved from the `purgatory` to the `learnt` buffer are simply incorporated into the learnt clauses database as if they were learnt by solvers themselves.

6.3   Results

The experimental evaluation is divided into four parts. First, we evaluate the different ingredients of AMPHAROS. Then, we study the scalability of our solver. Finally, we compare AMPHAROS to the state-of-the-art and study the impact of the *rscm* measure.

On the impact of each component. The benefit of the three optional components (tree decomposition (**Tree**), clauses (**C**) and unit literals exchange (**UL**)) of AMPHAROS has been studied experimentally. To this end, several versions of AMPHAROS have been executed on 64 cores. These experiments, reported in Fig. 4, show gradual improvements when each of these options was taken into account in a cumulative way. From the Fig. (4a) and the cactus plot (b) several observations can be made.

First, Fig. 4a shows that whatever the combination of options is used, AMPHAROS is more efficient when the tree decomposition is used (Tree sets to true in the first column). The versions working on the initial problem in a competitive way, which could be regarded as portfolio parallel SAT solvers (with (C) or without (none) clause sharing), solve systematically less instances than the others running on sub-problem obtained from cubes. This shows the importance of how AMPHAROS solves the intensification/diversification dilemma using a tree decomposition.

Second, results show the importance of exchanging information between solvers. AMPHAROS that does not exchange information systematically solved less problems than the others which share clauses or unit literals. When we

Tree	Exchanges	SAT	UNS	Total
Yes	C + UL	**49**	**25**	**74**
Yes	C	47	21	68
Yes	UL	47	18	65
Yes	None	41	15	56
No	C	43	6	49
No	None	44	6	50

<div align="center">(a) Overview table</div>
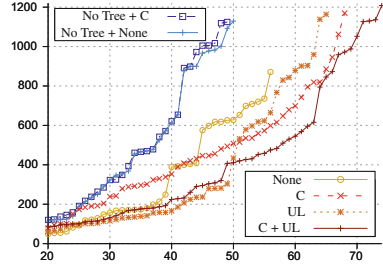
(b) Number of instances solved w.r.t. the time

Fig. 4. Comparing several versions of AMPHAROS on 64 cores. (a) gives the results of each version in term of solved instances. The columns represent, in that order, the fact that the tree decomposition is activated (**yes/no**), the kind of information exchanged (clauses (**C**) or/and unit literal (**UL**)), the number of SAT/UNSAT instances solved. (b) shows the number of solved instances (x-axis) w.r.t. the time (y-axis).

separately compare the two exchange options (C and UL), we observe that sharing clauses allows (as expected) to improve the solver on unsatisfiable problems. However, as pointed out in Fig. 4b, activating this option makes the solver slower on easy problems (solved with less than 600 s). This can be explained by the fact that the communication engendered to share clauses and manage them is significant and slows down the solvers on 'easy' problems.

Finally, as highlighted by these experiments, there is a synergy between the exchange options. Even if clause sharing drastically reduces the solver performance on easy benchmarks, combining this component with the unit literals allows one to deliver the most significant improvement in terms of number of successfully solved instances. From now, AMPHAROS is reported as the version using all components.

Scalability evaluation. To evaluate the scalability of AMPHAROS we run it on 8, 16, 32 and 64 cores. Figure 5 gives the number of solved instances w.r.t. the time by the different versions of AMPHAROS. It clearly demonstrates that our approach is highly scalable. The version running on 64 cores solves 49 SAT and 25 UNSAT benchmarks, that is 15 %, 45 % and 70 % more benchmarks than the one running on 32 (44 SAT and 20 UNSAT), 16 (36 SAT and 15 UNSAT) and 8 (33 SAT and 11 UNSAT) cores, respectively.
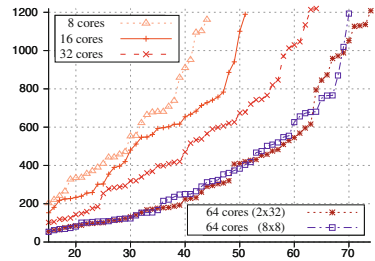
Fig. 5. Number of instances solved

In order to show the efficiency of our approach with more computers linked over the network, we also ran it with 64 cores using 8 computers with 8 cores each (see the curve 8 × 8 on Fig. 5.). This version solves 46 SAT and 24 UNSAT

benchmarks. Since we restrict the number of messages, we obtain similar results. Differences can be explain by the indeterminism of our approach.

AmPharoS versus the state-of-the-art. In order to evaluate AmPharoS with respect to existing work, we choose to compare our approach with the three best solvers of the last SAT competition that ran in the parallel track. These solvers are (in their rank order): Syrup [7], Treengeling and Plingeling [12]. Because they do not run with MPI, and we have no processor with 64 cores, we execute them on 32 cores. We also compare our solver on 64 cores with the work stealing parallel SAT solver Dolius [2] and the portfolio solver HordeSAT [8] (Fig. 6 reports results obtained).

Let us first consider the experiments launched on 32 cores. As reported in Fig. 6a, AmPharoS solves more instances than the other solvers. It is the best solver on the satisfiable benchmarks and solves the same number of unsatisfiable problems as Treengeling (which is also a divide and conquer based method). Comparing to Syrup and Plingeling, we can see that our solver significantly outperforms both on satisfiable problems but it is less efficient on unsatisfiable ones. This can be partially explained by the fact that AmPharoS essentially solves the unsatisfiable problems by totally refuting the tree (*i.e.* by closing all branches). Consequently, it seems that on 32 cores we do not have enough workers to achieve this goal within the time limit. When considering the running time of the solvers, reported in Fig. 6b, we can observe that AmPharoS is faster than Treengeling and Plingeling but it is slower than Syrup. This can be explained by the fact that Syrup solves several unsatisfiable problems in short time (the `6s` family for instance). If we consider the experiments run on 64 cores, we can see that our approach is highly competitive. AmPharoS is significantly better than Dolius and HordeSAT. Moreover, it is important to notice that during the competition Syrup (the winner of the parallel track) only used 32 cores instead of the 64 cores available. Consequently, it is possible to conclude that AmPharoS is more efficient than both Treengeling and Plingeling on 64 cores. More importantly, as reported in Fig. 6b, AmPharoS is very effective since it solves more instances and faster.

Solver	#thr.	SAT	UNS	Total
AmPharoS	32	**44**	20	**64**
Syrup	32	36	**26**	62
Treengeling	32	38	20	58
Plingeling	32	31	**26**	57
AmPharoS	64	**49**	25	**74**
HordeSAT	64	33	24	57
Dolius	64	33	17	50

(a) Overview table
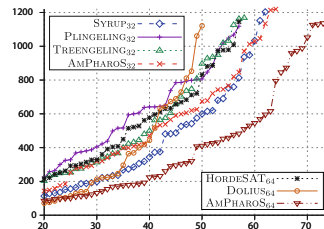
(b) Number of solved instances w.r.t. the time

Fig. 6. Comparing AmPharoS versus the state-of-the-art parallel solver. (a) gives results of each solver in term of solved instances w.r.t. the number of threads (**#thr.**).

Table 1. This table presents the obtained results on a representative set of benchmarks. Each line corresponds to an instance, with its satisfiability, identified by the leftmost column. The next four columns give the WC time (reported in seconds) to solve the instance w.r.t. the value of $rscm$ (static (set to 1, 3, 5 and 10) or dynamic (D)). The rightmost reports stastictics on the value obtained by the dynamic computation of $rscm$.

Benchmarks information		Time w.r.t. rscm					rscm Statistics			
Name	Sol.	1	3	5	10	D	Min	Max	Avg	Med
hitag2-10-60-0-65	UNS	563	173	120	127	304	1.20	2.36	2.11	2.28
jgiraldezlevy.109	UNS	544	243	214	154	260	1.60	4.32	3.76	4.12
minandmaxor128	UNS	788	IN	IN	IN	972	1.09	1.37	1.25	1.23
jgiraldezlevy.33	SAT	776	339	386	169	288	1.63	4.58	3.32	3.82
56bits-12.dimacs	SAT	114	168	180	395	101	1.25	1.60	1.43	1.46
004-80-8	SAT	248	412	16	264	110	1.41	4.64	3.10	3.09

Let us stress that none of these solvers are deterministic. To be fair, we ran all solvers just once and report the obtained results (as it was done in SAT competition 2015).

Impact of the *rscm* value. To conclude this section, we evaluate the impact of the $rscm$ value on our solver's performance. To this end, we selected a representative set of benchmarks and ran four versions of AMPHAROS with different values of $rscm$ (1, 3, 5 and 10) and compared them with the dynamically chosen value. Let us recall that $rscm$ has an impact on both the tree extension and the amount of exchanged clauses. Moreover, as mentioned in Sect. 5.2, the extending factor fe is fixed to 10 when $rscm > \sqrt[3]{100}$. Thus, the difference between $rscm = 5$ and $rscm = 10$ is only the amount of shared clauses. Table 1 shows that these instances do not have the same comportment with respect to the $rscm$ value. Some problems need to extend more (`jgiraldezlevy`) and others need to extend less and exchange more (`minandmaxor128`). It is also important to note some benchmarks are unpredictable (`004-80-8`). As regards the dynamic adjustment, we observe that it is in average often close to the best value.

7   Conclusion

We proposed a new parallel SAT solver, designed to work on many cores, based on the divide and conquer paradigm. Our solver allows two kinds of clause sharing, the classical one and one more linked to the division of the initial formula. Furthermore, we proposed to measure the degree of redundancy of the search by counting the number of subsumed shared clauses. With this measure, we are able to adjust dynamically the search, resulting in a new way of controlling the dilemma of intensification/diversification of the search. Experiments show promising results. Our main objective is to deploy a SAT solver among the cloud. Thus, this paper is a first step towards this goal and leads to many perspectives.

We plan to run our solver on a cloud architecture using grid computing. For that, we plan to run several MANAGER's in parallel letting solvers go from a MANAGER to another one. For that, we need to choose more carefully variables used for the division. Many possibilities arise like the notion of blocked literals recently used for SAT solving [14,27]. Finally, we also need to improve performances on unsatisfiable instances by paying more attention on shared clauses.

References

1. Audemard, G., Hoessen, B., Jabbour, S., Lagniez, J.-M., Piette, C.: Revisiting clause exchange in parallel SAT solving. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 200–213. Springer, Heidelberg (2012)
2. Audemard, G., Hoessen, B., Jabbour, S., Piette, C.: An effective distributed d&c approach for the satisfiability problem. In: 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014, Torino, Italy, pp. 183–187, 12–14 February 2014
3. Audemard, G., Lagniez, J.-M., Mazure, B., Saïs, L.: On freezing and reactivating learnt clauses. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 188–200. Springer, Heidelberg (2011)
4. Audemard, G., Lagniez, J.-M., Simon, L.: Improving glucose for incremental SAT solving with assumptions: application to MUS extraction. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 309–317. Springer, Heidelberg (2013)
5. Audemard, G., Lagniez, J.-M., Simon, L.: Just-in-time compilation of knowledge bases. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, 3–9 August 2013
6. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, pp. 399–404, 11–17 July 2009
7. Audemard, G., Simon, L.: Lazy clause exchange policy for parallel SAT solvers. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 197–205. Springer, Heidelberg (2014)
8. Balyo, T., Sanders, P., Sinz, C.: HordeSat: a massively parallel portfolio SAT solver. In: Heule, M. (ed.) SAT 2015. LNCS, vol. 9340, pp. 156–172. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24318-4_12
9. Belov, A., Lynce, I., Marques-Silva, J.: Towards efficient MUS extraction. AI Commun. **25**(2), 97–116 (2012)
10. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)
11. Biere, A.: Bounded model checking. In: Biere et al. [10], Chap. 14, vol. 185, pp. 455–481, February 2009
12. Biere, A.: Lingeling, Plingeling and Treengeling entering the SAT competition 2013. In: Proceedings of SAT Competition 2013, p. 51 (2013)
13. Biere, A.: Yet another local search solver and lingeling and friends entering the sat competition 2014. In: Proceedings of SAT Competition (2014)
14. Chen, J.: Minisat bcd and abcdsat: solvers based on blocked clause decomposition. In: SAT RACE 2015 Solvers Description (2015)

15. Chrabakh, W., Wolski, R.: The gridsat portal: a grid web-based portal for solving satisfiability problems using the national cyberinfrastructure. Concurrency Comput. Pract. Exp. **19**(6), 795–808 (2007)
16. Chu, G., Stuckey, P.J., Harwood, A.: Pminisat: a parallelization of minisat 2.0. Technical report, SAT Race (2008)
17. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
18. Feldman, Y., Dershowitz, N., Hanna, Z.: Parallel multithreaded satisfiability solver: design and implementation. Electron. Notes Theoret. Comput. Sci. **128**(3), 75–90 (2005)
19. Goldberg, E., Novikov, Y.: Berkmin: a fast and robust sat-solver. Discrete Appl. Math. **155**(12), 1549–1561 (2007)
20. Gomes, C.P., Selman, B., Kautz, H.A.: Boosting combinatorial search through randomization. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 1998, IAAI 1998, Madison, Wisconsin, USA, pp. 431–437, 26–30 July 1998
21. Guo, L., Hamadi, Y., Jabbour, S., Sais, L.: Diversification and intensification in parallel SAT solving. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 252–265. Springer, Heidelberg (2010)
22. Guo, L., Lagniez, J.-M.: Dynamic polarity adjustment in a parallel SAT solver. In: IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, pp. 67–73, 7–9 November 2011
23. Hamadi, Y., Jabbour, S., Sais, L.: Control-based clause sharing in parallel SAT solving. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, pp. 499–504, 11–17 July 2009
24. Hamadi, Y., Jabbour, S., Sais, L.: Manysat: a parallel SAT solver. JSAT **6**(4), 245–262 (2009)
25. Heule, M.J.H., Kullmann, O., Wieringa, S., Biere, A.: Cube and conquer: guiding CDCL SAT solvers by lookaheads. In: Eder, K., Lourenço, J., Shehory, O. (eds.) HVC 2011. LNCS, vol. 7261, pp. 50–65. Springer, Heidelberg (2012)
26. Hyvärinen, A.E.J., Junttila, T., Niemelä, I.: Partitioning SAT instances for distributed solving. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 372–386. Springer, Heidelberg (2010)
27. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
28. Lagniez, J.-M., Biere, A.: Factoring out assumptions to speed up MUS extraction. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 276–292. Springer, Heidelberg (2013)
29. Le Berre, D.: Exploiting the real power of unit propagation lookahead. Electron. Notes Discrete Math. **9**, 59–80 (2001)
30. Marques-Silva, J., Sakallah, K.: GRASP - a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, pp. 220–227 (1996)
31. Martins, R., Manquinho, V.M., Lynce, I.: Improving search space splitting for parallel SAT solving. In: 22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, vol. 1, pp. 336–343, 27–29 October 2010
32. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, pp. 530–535. ACM, 18–22 June 2001

33. Nadel, A., Ryvchin, V.: Efficient SAT solving under assumptions. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 242–255. Springer, Heidelberg (2012)

34. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 294–299. Springer, Heidelberg (2007)

35. Rintanen, J.: Planning and SAT. In: Biere et al. [10], Chap. 15, vol. 185, pp. 483–504, February 2009

36. Roussel, O.: ppfolio. http://www.cril.univ-artois.fr/~roussel/ppfolio

37. SAT-race (2015). http://baldur.iti.kit.edu/sat-race-2015/

38. Schubert, T., Lewis, M., Becker, B.: Pamiraxt: parallel SAT solving with threads and message passing. J. Satisfiability Boolean Model. Comput. **6**(4), 203–222 (2009)

39. Semenov, A., Zaikin, O.: Using monte carlo method for searching partitionings of hard variants of boolean satisfiability problem. In: Malyshkin, V. (ed.) PaCT 2015. LNCS, vol. 9251, pp. 222–230. Springer, Heidelberg (2015)

40. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 244–257. Springer, Heidelberg (2009)

41. van der Tak, P., Heule, M.J.H., Biere, A.: Concurrent cube-and-conquer. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 475–476. Springer, Heidelberg (2012)

42. Zhang, H., Bonacina, M.P., Hsiang, J.: Psato: a distributed propositional prover and its application to quasigroup problems. J. Symbolic Comput. **21**(4), 543–560 (1996)

Improved Linearization of Constraint Programming Models

Gleb Belov[1(✉)], Peter J. Stuckey[2], Guido Tack[1], and Mark Wallace[1]

[1] Monash University, Caulfield Campus, Caulfield East, Australia
{gleb.belov,guido.tack,mark.wallace}@monash.edu
[2] Data61, CSIRO, University of Melbourne, Parkville, Australia
pstuckey@unimelb.edu.au

Abstract. Constraint Programming (CP) standardizes many specialized "global constraints" allowing high-level modelling of combinatorial optimization and feasibility problems. Current Mixed-Integer Linear Programming (MIP) technology lacks both a modelling language and a solving mechanism based on high-level constraints.

MiniZinc is a solver-independent CP modelling language. The solver interface works by translating a MiniZinc model into the simpler language FlatZinc. A specific solver can provide its own redefinition library of MiniZinc constraints.

This paper describes improvements to the redefinitions for MIP solvers and to the compiler front-end. We discuss known and new translation methods, in particular we introduce a coordinated decomposition for domain constraints. The redefinition library is tested on the benchmarks of the MiniZinc Challenges 2012–2015. Experiments show that the two solving paradigms have rather diverse sets of strengths and weaknesses. We believe this is an important step for modelling languages. It illustrates that the high-level approach of recognizing and naming combinatorial substructure and using this to define a model, common to CP modellers, is equally applicable to those wishing to use MIP solving technology. It also makes the goal of solver-independent modelling one step closer. At least for prototyping, the new front-end frees the modeller from considering the solving technology, extracting very good performance from MIP solvers for high-level CP-style MiniZinc models.

Keywords: Combinatorial optimization · High-level modelling · Automatic reformulation · Linear decomposition · Context-aware reformulation

1 Introduction

Constraint Programming (CP) operates in terms of specialized constraints, from basic ones such as arithmetic, to high-level "global constraints" [3], and their filtering/explanation algorithms. A solver which handles a model's high-level structure in terms of global constraints, can take advantage of this knowledge in different ways. When the solver does not provide a handler for a certain constraint, the latter can be expressed by more basic entities.

Current Mixed-Integer Programming (MIP) technology lacks a modelling language based on global constraints, so that a dedicated MIP modeller has to hard-code a chosen MIP decomposition of his real problem. Moreover, a MIP solver might need to reverse-engineer the high-level model information for efficiency [24].

We consider automatic linearization of CP models, producing good quality MIPs. MiniZinc [21] is a solver-independent Constraint Programming modelling language. A MiniZinc model is translated into FlatZinc, a low-level language, and the translation is controlled by a redefinition library. The MiniZinc front-end is now supported by some 20 solvers, including finite domain solvers, SAT solvers, Lazy Clause Generation solvers, and even local search solvers [4]. Annual MiniZinc competitions [27] provide a basis for comparing solvers and exploring their strengths and weaknesses.

A number of modelling front-ends are available for MIP solvers, including GAMS [2] and AMPL [10] which focus on general Mathematical Programming. For combinatorial problems, AMPL supports logical constraints and counters. A similar functionality is offered by ZIMPL [15]. There are a number of CP languages offering automatic translation to MIP [1].

Our vision is that MiniZinc becomes an accepted and even widely-used modelling language within the OR community, thus helping to narrow the divide between OR, CP and SAT researchers, and to simplify prototyping. To this purpose we seek to ensure that pure MIP models, when formulated in MiniZinc, have similar performance to AMPL and GAMS. This requires no reformulation, but care needs to be taken in user and solver interfaces. Beyond that, we try to optimize the MIP-compatible reformulation of CP models to make it flexible and extensible. This enables the modeller to use the CP style modelling where combinatorial substructure is captured using global constraints, and obtain good performance for their problem, using state-of-the-art MIP, CP and SAT solving technologies on the same models.

In the MiniZinc Challenge [27] MIP solvers have had some success, but MIP did not appear competitive on most of the Challenge benchmarks. Some models are inherently more efficient for MIP solving, e.g., assignment problems (see the example in Sect. 2.2), and problems involving network flow. But we were suspicious that the relatively poor performance of MIP solvers was an artifact of a naive transformation of CP models to MIP. By improving linearization we can see the true potential of MIP solvers on the Challenge benchmarks.

In automatic reformulation, it is up to the modelling system to provide efficient translation for the target solver. To yield efficient transformed models it is important to ensure that auxiliary variables generated during reformulation are not unnecessarily duplicated [23]. Cire et al. [7] define an interactive system that aids automatic detection of equivalent auxiliary variables produced in reformulations of various parts of a model. MiniZinc 2.0 takes a different approach through user-defined functions [28] which are used to avoid duplication in the first place.

Refalo [23] presents a system for automatic reformulation of global constraints into MIPs. He observes that the reformulations are usually standard and tight. The system supports dynamic reformulation: as more information about the model becomes available during solving, the reformulation is updated. However, the implementation is bound to a hybrid of specific CP and MIP solvers.

Among the "basic" non-linear constraints we consider domain constraints, restricting an integer variable to take values in a specified set, both in static as well as reified version (i.e., depending on another condition). Such constraints can appear on their own in a model, or be produced by the decomposition of other non-linear constraints, such as disjunction, array element access, and many others. We propose a coordinated decomposition of domain constraints which takes into account all those of a group of dependent variables.

The next section gives introductory examples. Section 3 discusses general linearization methods, in particular introducing the new domain constraint decomposition. Experimental results follow.

2   Basics and Redefinition Examples

This section provides an overview of MiniZinc's redefinition mechanism and some motivating examples.

2.1   Basics on MiniZinc and Solver-Specific Redefinitions

MiniZinc [28] is a declarative modelling language. It builds constraint structures using *predicates*, here is a toy example:

```
1  predicate small(int: m, var int:y) =  -m <= y /\ y <= m;
2  predicate p(int: u, var bool: b, var int: x) =
3          (b <-> small(u,x));
4  constraint p(4,false,v);
```

Global constraints [3], such as the well known `alldifferent`, are also specified as predicates.

When the model is compiled for a specific solver, the front-end looks for a solver-specific redefinition of the global constraints used. If none is provided, MiniZinc has a default decomposition, for example the standard library definition for `alldifferent` is:

```
1  predicate alldifferent(array[int] of var int: x) =
2        forall(i,j in index_set(x) where i < j)
3          ( x[i] != x[j] );
```

However the solver can provide its own redefinition. In particular, it can forward the predicate call unchanged and use specialized algorithms. The translated model is converted to the low-level FlatZinc format and can be passed to the solver, or used directly if the solver is linked in the same executable.

For example, the `alldifferent` constraint can be redefined in a different way than given above for a linear solver:

```
1  predicate alldifferent(array [Set1] of var Set2: x)  =
2      forall (j in Set2)( sum(i in Set1)(x[i]==j) <= 1 );
```

This redefinition automatically introduces an auxiliary zero-one variable to encode the assignment of a variable to a value, `x[i]==j`, see Sect. 3. Note that this auxiliary variable is *re-used* whenever this equality is encoded again, see Sect. 3.1.

The redefinition library for MIP is located in folder `share/minizinc/linear` of the MiniZinc distribution. To use this library, the `mzn2fzn` compiler is called with options `-G linear`. In particular, files `redefinitions*.mzn` re-define the basic constraints, such as logical connectives and min/max. Most global constraints are specified in dedicated files, for example `lex_less.mzn`. If a library does not provide a header for some global, its default decomposition is taken from the standard library `share/minizinc/std`.

2.2   Linearization Example: Assignment Problem

Consider an assignment problem. Its natural CP model is:

```
1  set of int: WORKER ;   % workers
2  set of int: TASK ;     % tasks to be assigned to workers
3  array[WORKER,TASK] of int: value;
4  array[WORKER] of var TASK: task; % which task worked on by each worker
5  include "alldifferent.mzn";
6  constraint alldifferent(task); % each worker works on a different task
7  solve maximize sum(w in WORKER)(value[w,task[w]]);
```

The natural MIP formulation of the model is the following one:

```
1  set of int: WORKER ;   % workers
2  set of int: TASK ;     % tasks to be assigned to workers
3  array[WORKER,TASK] of int: value;
4  array[WORKER,TASK] of var 0..1: worker_task;
5  constraint forall(w in WORKER)          % one task per worker
6                (sum(t in TASK)(worker_task[w,t]) = 1);
7  constraint forall(t in TASK)
8                (sum(w in WORKER)(worker_task[w,t]) <= 1); % alldifferent
9  solve maximize sum(w in WORKER, t in TASK)(value[w,t] * worker_task[w,t]);
```

Unsurprisingly the MIP solver is effectively "infinitely" faster than a CP solver on this problem since the MIP solver will effectively implement a polynomial-time maximal matching algorithm using the linear integer constraints that arise in its formulation. The challenge for the automatic linearization is to ensure that the "natural" CP model above results in the MIP formulation being sent to the MIP solver, so that we can make use of the insights of combinatorial substructure without being penalized. This is particularly important when we want to solve assignment problems with other side constraints.

As explained in Sect. 2.1, our automatic linearization of `alldifferent` produces the exact equivalent of its "natural" MIP decomposition. For the objective function, which accesses element `task[w]` in each row `w` of matrix `value`, the compiler transforms nested matrix access `value[w,task[w]]` into a standard array access represented by the global constraint `element` [3]. The latter is linearized as follows [13]:

```
1  sum (t in TASK) ( value[w,t] * (task[w]==t) )
```

Note that the auxiliary binary variable representing the equality `task[i]==j` is re-used, which altogether gives the natural MIP formulation. There will be some overhead for the (now useless) original `task` variables. However we have an instance of a 3D orthogonal packing model where such variables improve search behavior of IBM ILOG CPLEX 12.6.3 [14].

2.3   Linearization Example: Tour Guide Allocation Problem

An application brought to us by a local company is the tour guide allocation problem. For a set of planned tours with fixed locations and times, the requirement is to minimize the total number of guides needed as well as the travel costs of the guides between their tours.

Let matrix `travel_cost` contain the travel costs between tours and a 4-column matrix `tour` contain start day, duration, start and end location of a tour in each row. Variable array `succ` describes the successors of each tour in its guide's sequence of tours, as follows:

```
1  int: tour_ct;          % The total number of planned tours
2  set of int: C = 1..4;  % Columns of tour data structure
3  int: SDay = 1; int: Dur = 2; int: SLoc = 3; int: ELoc = 4; % Column names
4  array [1..tour_ct,C] of int: tour;
5  int: loc_ct;           % Number of locations
6  array [1..loc_ct, 1..loc_ct] of int: travel_cost;
7  array [1..tour_ct-1] of var 1..tour_ct: succ;
```

The last tour with index `tour_ct` is the END tour with a zero distance to all other tours' locations. This ensures that in an optimum no two tours have the same successor (different from END).

The total travel cost is the sum of the cost of traveling from the end of each tour to the start of its successor (as recorded by `succ`):

```
1  constraint    total_travel_cost = sum (t in 1..tour_ct-1)
2                     (travel_cost[tour[t,ELoc],tour[succ[t],SLoc]]);
```

As in Sect. 2.2, the nested matrix element accesses are simplified by the compiler, resulting in a linear constraint.

Another array of decision variables is `first_tour`. It tells us how many tour guides are to be used. It does this by selecting certain tours to be the first tour on (some) tour guide's sequence of duties. Then, every tour must have a tour guide (either it must be a first tour or it must be the successor of another tour):

```
1  array [1..tour_ct-1] of var bool: first_tour;
2  constraint forall(t in 1..tour_ct-1)
3                 ( first_tour[t] \/
4                 ( exists (t2 in 1..tour_ct-1) (t = succ[t2]) )
5                 ) ;
```

Again, decisions `t==succ[t2]` are converted into auxiliary binary variables, using either unary decomposition or domain refinement (Sect. 3). These auxiliary variables are automatically the same as in the linearization of the travel cost.

Finally, the successor of tour `t` must have a start date greater than or equal to the start date of `t` plus the duration of `t`:

```
1  constraint forall(t in 1..tour_ct-1)
2               (tour[t,SDay]+tour[t,Dur] <= tour[succ[t],SDay]);
```

The "every tour must have exactly one tour guide" constraint can be made explicit by a direct MIP-tailored network flow-type formulation as follows:

```
1  constraint forall(t in 1..tour_ct-1)
2               (first_tour[t] + sum(t2 in 1..tour_ct-1)(t = succ[t2]) =
   1);
```

On an example with 25 locations and 41 tours, CP finds only a suboptimal solution in observable time. Previously it was necessary to write a different MiniZinc model to elicit the efficient performance of a network-flow model with a MIP solver. Now, with automatic linearization, with or without the "every tour has exactly one tour guide" constraint, IBM ILOG CPLEX 12.6.3 [14] proves an optimum without branching.

3   Linearization

This section discusses some basic linearization principles, introduces domain refinement, and discusses decomposition of the most commonly used global constraints.

3.1   Linearization Principles

Linearization by "Big-Ms". The basic linearization method for complex constraints is the so-called big-M transformation (see e.g. [13,19]). Given a linear constraint $e \leq 0$ in disjunction with a Boolean b, that is $e \leq 0 \vee b$ or equivalently $\neg b \rightarrow e \leq 0$, then if M is the largest possible value linear expression e can take, this can be expressed using the linear constraint $e \leq Mb$.

For example, $x \neq y$ is equivalent to a disjunction between two inequalities:

$$x \geq y + 1 \ \vee \ y \geq x + 1 \tag{1}$$

which can in turn be transformed by introducing a binary variable b into the conjunction of two implications: $b \rightarrow x \geq y + 1$ and $\neg b \rightarrow y \geq x + 1$, which can then be transformed to linear constraints. Assume x and y range over $[0, 10]$ we can encode the first constraint using the linear constraint $y + 1 - x \leq 11(1 - b)$ and the second by $x + 1 - y \leq 11b$.

Linearization of complex constraints consists of breaking them down into reified linear constraints, and then replacing these with linear constraints using the big-M method illustrated above, or other methods described in this section. For space reasons we don't describe the MIP decompositions of other basic constraints, such as logical ones, referring the reader, e.g., to [13,23].

Example 1. Consider the model on the left in Fig. 1. Using "big-M"s, we can linearize the two constraints as shown on the right in the same figure. Its continuous relaxation allows the solution `x==5.5; beta1==beta2==0.75`.   □

```
1  var 0..10: x;
2  var bool:  beta1;
3  var bool:  beta2;
4  constraint beta1 <-> x<=4;
5  constraint beta2 <-> x>=7;
```

```
1  x-4 <= 6*(1-beta1)
2  5-x <= 5*beta1
3  7-x <= 7*(1-beta2)
4  x-6 <= 4*beta2
```

Fig. 1. Example model and its "big-M" linearization

Linearization with Unary Encoding of the Domain. An alternative approach to linearization of complex constaints is to introduce a binary variable b_k^x for each value k in the domain $D(x)$ of x [23]. The correspondence between the binary variables and the original integer variable can be enforced by the linear constraints

$$\sum_{k \in D(x)} b_k^x = 1, \tag{2a}$$

$$\sum_{k \in D(x)} k b_k^x = x. \tag{2b}$$

Unary encoding introduces a lot of auxilliary variables, however it is usually preferred due to its tighter continuous relaxation. There are many constraints which are best transformed using these binary variables, including `alldifferent`, `element` (see [23] and Sect. 2.2), `inverse`, multiplication of variables, and some others.

Tight Reformulation Using Common Subexpression Elimination. To achieve a tight MIP model without duplicate variables and constraints, it is essential that when a constraint on the same variable is transformed using its unary encoding, the same binaries are used. When the translation is controlled by a library, this can be achieved automatically through MiniZinc's mechanism of user-definable functions [28].

To introduce these binaries, we use the function `eq_encode(var int: x)` (which was named `int2array` in [28]), returning an array of 0–1 variables and imposing linear constraints (2). Now every time this function is invoked on a variable x, MiniZinc's common subexpression elimination ensures that the same binaries are reused, even if the function is embedded in a predicate or another function.

However there still can be information loss. For example, $x \neq y - 5$ or $y \neq x$ would be linearized using unary encodings of variables $z' = x - y + 5$ and $z'' = y - x$, respectively. The current capabilities of the MiniZinc language do not allow it to recognize that we could make use of the same unary encoding for these cases and we tackle this issue together with unified domain refinement in Sect. 3.2.

Multiplication. In MiniZinc 1.6, the decomposition for FlatZinc predicate `int_times` constraining $z = xy$ was $z = (xy_{\min}, \ldots, xy_{\max})_{y-y_{\min}+1}$, or equivalently, using explicit calls to the global constraint `element` [3], `element`

$(y - y_{\min} + 1, [xy_{\min}, \ldots, xy_{\max}], z)$, where $y_{\min}, y_{\max}$ are the finite bounds of y. Note this method will also work when x is real-valued.

In the cases of a small (chosen as 4..20) product domain size $|D(x)| \times |D(y)|$ and no variable domain having the form $\{0, k\}, k \in \mathbb{Z} \setminus \{0\}$, experiments proved that it is advantageous to use the following alternative encoding:

$$z = \sum_{i,j} i \times j \times b_{ij}^{xy}, \qquad \text{where} \quad b_{ij}^{xy} = 1 \leftrightarrow (b_i^x = 1 \wedge b_j^y = 1). \qquad (3)$$

If $|D(x)| = |D(y)| = 2$ and $0 \in D(x) \cap D(y)$, we apply Boolean conjunction instead. All these decompositions seem reasonably strong because experimentation with McCormick envelopes [18] did not show better results.

3.2   Linearization of Domain Constraints

A critical class of constraint for linearization are the so called domain constraints. Under *domain constraint* for variable $x \in \mathbb{Z}$ we understand any of the following:

$$x \in S, \qquad (4a)$$

$$\beta \leftrightarrow x \in S, \qquad (4b)$$

where $S \subset \mathbb{Z}$ is a finite integer set and β a Boolean variable. (4a) is a static and (4b) is a reified domain constraint. In FlatZinc they are imposed by predicates `set_in(_reif)`.

This class of constraints generalizes some other non-linear constraints, such as comparisons with a constant: $x \neq a$ (static and reified, `int_ne(_reif)`), $x = a$ and $x \leq a$ (reified, `int_(eq/le)_reif`). Of the two latter, only reified versions are non-linear. W.l.o.g., FlatZinc doesn't consider other comparison operations as they can be reduced to "$\leq$" by variable substitution.

Moreover, comparisons between two variables $x, y \in \mathbb{Z}$ can be transformed to constraints (4a), (4b) by introducing a variable for their difference: $z = x - y$. Then, for example, $x \neq y$ is equivalent to $z \neq 0$.

Domain constraints (4a) and (4b) can be straightforwardly linearized using the unary encoding (2) by the following constraints (5a) and (5b), respectively:

$$1 = \sum_{k \in S} b_k^x, \qquad (5a)$$

$$\beta = \sum_{k \in S} b_k^x. \qquad (5b)$$

Domain Refinement for Integer Variables. The unary encoding (2) can introduce a lot of auxilliary variables b_k^x for x with a big domain $D(x) \subset \mathbb{Z}$. We propose a refined domain structure as follows. Let $x \in S$ be a constraint of the form (4a). Let the following list:

$$SL(S) = \arg\min \left\{ n \mid S = \mathbb{Z} \cap \bigcup_{i=1}^{n} [l_i, u_i], \ l_i, u_i \in S, \ i = \overline{1, n} \right\} \qquad (6)$$

be the smallest list of integer-bounded intervals covering S and including no other integer values. Then, $x \in S$ is equivalent to the following system:

$$\sum_i l_i \tilde{b}_i \leq x \leq \sum_i u_i \tilde{b}_i, \tag{7a}$$

$$\sum_i \tilde{b}_i = 1, \tag{7b}$$

$$\tilde{b}_i \in \{0, 1\}, \qquad i = \overline{1, |SL(S)|}. \tag{7c}$$

System (7) generalizes the unary encoding (2). Note that when the full unary encoding `eq_encode(x)` is already introduced for a given variable x, the MiniZinc transformation ensures that it is used for the domain constraints instead of the above. This prevents both systems (2) and (7) from being present in the model.

System (7) can be seen as a disjunction of 1D polyhedra. However we are not aware of any previous results in line with the unified domain refinement introduced below.

Unified Domain Refinement. We propose a single domain refinement which can be used to decompose all the domain constraints on a given variable, as well as those on dependent variables.

Example 2. (continued from Example 1). For the model of Fig. 1, consider the following list of intervals:

$$\overline{SL} = ([0, 4], [5, 6], [7, 10]),$$

and the corresponding system (7). Then we can linearize the model by imposing the following equivalences:

$$\beta_1 = \tilde{b}_1, \tag{8a}$$

$$\beta_2 = \tilde{b}_3. \tag{8b}$$

The solution `beta1 = beta2 = 0.75; x == 5.5;` of Example 1 is no longer feasible, in the linear relaxation, simply by (7b).   □

W.l.o.g., for a given variable x we have exactly one static domain constraint (4a) (with $S = D(x)$) and possibly several reified constraints (4b).

Definition 1. Given an integer variable x and all its domain constraints

$$x \in D(x), \tag{9a}$$

$$\beta_j \; \leftrightarrow \; x \in S_j, \qquad j \in J_x, \tag{9b}$$

define the *unified domain refinement* $\overline{SL}_x$ as the list of the isolated intervals of the set

$$\mathcal{S}_x = SL(D(x)) \cap \bigcap_{j \in J_x} \left(SL(S_j) \cup SL(D(x) \setminus S_j) \right). \tag{10}$$

Note that unary encoding (2) represents a special case of domain refinement, namely it is equivalent to system (7) based on the degenerate interval list $([l_k, u_k] \mid l_k = u_k = k, \; k \in D(x))$.

Theorem 1. *For an integer variable x, system (7) based on the interval list $\overline{SL}_x$ correctly linearizes the static constraint (9a). For each $j \in J_x$, the reified constraint (9b) is correctly linearized by the following equation:*

$$\beta_j = \sum \left\{ \tilde{b}_i \mid l_i, u_i \in S_j, \; i \in \left\{1, \ldots, |\overline{SL}_x|\right\} \right\}. \tag{11}$$

Proof. Note that $\mathcal{S}_x \cap \mathbb{Z} = D(x)$, which proves correctness for (9a). The sublist $([l_i, u_i] \mid l_i, u_i \in S_j)$ of $\overline{SL}_x$ covers all elements of $S_j \cap D(x)$, proving (11). □

Theorem 2. *For an integer variable x with domain constraints (9), the continuous relaxation of the decompositions (7), (11) based on unified domain refinement $\overline{SL}_x$ as well as that of unary encoding (2), (5b) are equally strong in terms of the high-level variables (x and β_j, $j \in J_x$).*

Proof. Given a continuous solution $\left(x, \beta_j|_{j \in J_x}, b_k^x|_{k \in D(x)}\right)$ of unary encoding, it is easy to see that $\left(\tilde{b}_i = \sum_{k=l_i}^{u_i} b_k^x\right)|_{i=1}^{|\overline{SL}_x|}$ fulfills (7) and (11).

Vice versa, given a continuous solution $\left(x, \beta_j|_{j \in J_x}, \tilde{b}_i|_{i=1}^{|\overline{SL}_x|}\right)$ of (7), (11), set

$$r = \frac{x - \sum_i l_i \tilde{b}_i}{\sum_i u_i \tilde{b}_i - \sum_i l_i \tilde{b}_i} \in [0, 1].$$

For each $i \in \left\{1, \ldots, |\overline{SL}_x|\right\}$, if $l_i = u_i$ then set $b_{l_i}^x = \tilde{b}_i$, otherwise select $(b_k^x)|_{k=l_i}^{u_i}$ so that $\tilde{b}_i = \sum_{k=l_i}^{u_i} b_k^x$ (which provides (2a) and (5b)) and

$$\left(\sum_{k=l_i}^{u_i} k \frac{b_k^x}{\tilde{b}_i} - l_i\right) \Big/ (u_i - l_i) = r,$$

which is in general non-unique. This fulfills (2b):

$$\sum_{k \in D(x)} k b_k^x = \sum_{i=1}^{|\overline{SL}_x|} \sum_{k=l_i}^{u_i} k b_k^x = \sum_{i=1}^{|\overline{SL}_x|} \tilde{b}_i (r(u_i - l_i) + l_i) = x. \qquad \square$$

We see that unary encoding can have non-unique equivalents for a solution of unified refinement, leading to symmetries.

Dependent Variables. When there is a set of variables $\{x_1, \ldots, x_n\}$ that are pairwise linearly dependent (i.e. $\forall 1 \leq i < j \leq n \exists a_{ij} b_{ij}$ s.t. $x_i = a_{ij} x_j + b_{ij}$), if at least one of them has a unary encoding generated by a specialized global, it can be re-used for the domain constraints of all $\{x_i\}$. Otherwise, all the domain constraints on $\{x_i\}$ can be projected onto just one of them, using a single unified domain refinement.

The unification procedure was implemented as a post-processing step in the MiniZinc compiler v2.0.10 but still controllable from the redefinition library. It looks for linearly dependent variables in several ways, for example if two variables are initialized by linear expressions whose non-constant parts are multiples of each other. This occurs for various auxiliary variables introduced in reformulations.

3.3   Global Constraint Decompositions

The MiniZinc distribution defines default decompositions for over 100 global constraints. The reformulations described in Sect. 3.2 above ensure that most of these default decompositions can be directly re-used for MIP, producing tight linear reformulations of the global constraints, where duplication of auxiliary variables has been automatically minimised.

For a few constraints where default decomposition is not MIP-efficient, we have implemented tailored MIP formulations, listed in the directory `share/minizinc/linear` of the MiniZinc distribution.

alldifferent, inverse, alldifferent_except_0: We have already seen how `alldifferent` is linearised using the unary encoding. One can linearise `inverse` similarly. The constraint `alldifferent_except_0` is a simple variation of `alldifferent` and much more pleasing than the constraint programming decomposition:

```
1  predicate alldifferent_except_0(array [Set1] of var Set2: x)  =
2      forall (j in Set2 diff {0})( sum(i in Set1)(x[i]==j) <= 1 );
```

element, table: We have already seen how `element` is linearised using the unary encoding. The table constraint $\texttt{table}([x_1, \ldots, x_n], T)$ is encoded by defining auxiliary 01 variables $\lambda_i, 1 \leq i \leq m$ for each of the m rows in the table and then equating $x_j = \sum_{i=1}^{m} \lambda_i T_{ij}$. This is a direct extension of the `element` encoding, minus the index element.

cumulative: The global `cumulative` constraint, limiting the total amount of a renewable resource available to all tasks at any moment of time, is frequent in scheduling problems [25]. It can be used to express `alldifferent`, as in the `ghoulomb.mzn` benchmark, and as a redundant constraint in packing problems [26].

Two forms of reasoning used in the cumulative constraints are reasoning about the ordering of tasks ("task decomposition") [25], and reasoning about the tasks running at each time slot ("time decomposition") [12].

Transforming the cumulative constraint for MIP can be costly in terms of both the number of variables and constraints. While the number of variables resulting from the task decomposition is proportional to the number of tasks squared, the time decomposition ultimately requires a variable for each task

indicating its relation to each time slot, which requires a number of variables proportional to the product of tasks and time slots.

The time decomposition of `cumulative` is currently the default in MiniZinc, and thus was solely used by the previous linearization library. We found that when the product of the number of time slots and the number of tasks exceeds a certain parameter (chosen as 2000), it is advantageous to use the task decomposition of `cumulative` and not the time decomposition.

`circuit` and `subcircuit`: The global constraints `circuit` and `subcircuit` take an argument vector x, where `x[i]` denotes the successor of node `i` (or just `i` if it is not included in the subcircuit). They ensure that there are no separate cycles and each node is in exactly (for `subcircuit`, in at most) one loop.

The previous linearization library had no special translation for them, resulting in the usage of standard decompositions. As an example, for subcircuit they involved ordering constraints of the type

$(\langle ordering\ condition \rangle)$ `-> order[x[i]] = order[i] + 1`

where auxiliary variable `order[i]` is the order of node `i` in the subcircuit, starting from the least-index node. The order of excluded nodes is not constrained. Expression `order[x[i]]` is a variable subscript (flattened as predicate `array_var_int_element`) and hard to linearize efficiently.

These globals are now encoded as variants of the Miller-Tucker-Zemlin formulation [20]. Interestingly, in an experiment with the lifted MTZ cuts of Desrochers and Laporte [9], we observed inferior behaviour when we tested them using IBM ILOG CPLEX 12.6.1 [14].

`regular`: Probably the most difficult global constraint for the previous linearization library is `regular`. It requires that the sequence of values in the control vector x satisfies a deterministic finite automation defined by the acceptable states vector a and a transition function d mapping the current state and the control value into the next state: `a[i+1] = d[a[i], x[i]]`. The default decomposition just uses the latter prescription directly, resulting in a series of `element`'s.

Specialized propagation algorithms for `regular`, cf. [8], construct the graph of achievable/feasible states for each step, called *layered graph*. Its nodes correspond to the unary encodings of the state variables `a[i]` for each step i: node (i, k) means `a[i]==k`, and arcs denote the transitions between the nodes of the consecutive steps (layers). A network-flow approach in [8] uses such a graph, implemented by an external procedure, and formulates the network-flow constraints in a MIP-typical way, namely with binary variables $\lambda_{(i,k_1),(i+1,k_2)} \in \{0,1\}$ for the flow on each arc $((i,k_1),(i+1,k_2))$.

We implemented this reformulation in MiniZinc, iteratively tightening the domains of the state variables `a[i]` and introducing the above-mentioned arc flow variables.

4   Experiment

To validate the MIP reformulations described above, we tested them on leading commercial and free MIP solvers, and compared them with the best solvers based on results from the MiniZinc Challenge.

As test instances we used 400 instances from MiniZinc Challenges 2012–2015. Naturally these instances are advantageous for the solvers proven on the very same test set!

The MIP solvers we tested were:

– commercial solver Gurobi 6.5.1 [11],
– commercial solver IBM ILOG CPLEX 12.6.3 [14],
– free solver COIN-OR Branch-and-Cut (CBC) 2.9.8 [17].

We tested the MIP solvers each under three configurations: default (with all linearization approaches), "no DR" (without domain refinement, Sect. 3.2), and "old" (with the old linearization library from MiniZinc 1.6 however supplemented with MIP-tailored globals `alldifferent`, `table`, and `inverse`, Sects. 2 and 3). The multi-pass compilation of models suggested in [16] was not considered as it currently fails on 15 instances.

The best solvers from the MiniZinc Challenge we used for comparison were:

– Opturion CPX [22], overall official winner of the Challenges 2013 and 2015,
– Chuffed [5,6], not prize-eligible in the Challenge.

For these solvers we used the search strategy specified by the model.

All solvers were executed sequentially (1 thread) on an Intel i7-4771 CPU @ 3.50 GHz with a memory limit of 12 GB per process. MiniZinc 2.0.13[1] was used to flatten the models. The actual FlatZinc-to-solver interfaces are going to be released as part of the upcoming MiniZinc 2.1. Solving time was limited to 5 min total CPU time per method/instance. Flattening time was not limited.

In Table 1 we report the following data for each solver and configuration: the number of optimal (opt), feasible but not optimal (feas), satisfied (sat), proven infeasible (inf), not flattened (nofzn), failed (fail) (solver crashed or did not stop normally in 500 s), and other cases (other) — where none of the previous results were achieved. i.e., the solver ran without finding feasible solutions and did not crash.

1. Note from Table 1 that sometimes the MiniZinc-to-FlatZinc compilation cannot produce a working model (columns `nofzn` and `fail`). For MIP, the main causes were the globals `cumulative`, `regular` and `table` involving variables with large domains, leading to huge decompositions and thus to big MIP models where MIP solvers run out of memory or just stop responding. The same happens to the CP solvers when they don't handle a global constraint directly and it has to be decomposed, in this case `cumulative` with variable durations and resource demands in 2 instances of `mznc2013/fjsp`.

---

[1] minizinc.org.

Table 1. Comparison of solvers and configurations

Solver+config	opt	feas	sat	inf	nofzn	fail	other
Gurobi	160	113	48	3	5	1	70
Gurobi, noDR	157	115	45	2	5	2	74
Gurobi, old	133	118	28	5	16	4	96
CPLEX	139	121	46	2	5	1	86
CPLEX, noDR	141	124	47	3	5	0	80
CPLEX, old	124	118	26	4	16	2	110
CBC	86	82	24	2	5	32	169
CBC, noDR	78	69	20	1	5	40	187
CBC, old	61	58	8	1	16	51	205
Chuffed	157	142	54	5	2	0	40
Opturion CPX	130	159	37	5	2	0	67

2. The linear solvers successfully find and prove optimality - Gurobi beats both Chuffed and Opturion CPX in terms of number of optimal solutions.
3. The new domain refinement is definitely beneficial to Gurobi and more so for CBC, while strangely it is disadvantageous for CPLEX. We believe it may interfere with some presolve simplifications in CPLEX.

Even the free MIP solver CBC, with helpful support from its developers, now runs bug-free and gives results on nearly half the instances. Moreover, as revealed in Table 2 CBC sometimes succeeds where the Challenge solvers fail.

In Table 2 we present pairwise set difference analysis for the solvers' best configurations. For each comparison of two solvers/configurations, we report the following numbers: `Oopt` is the number of cases where only that solver proved optimality (and the other solver had at best feasibility); `Ofea` is the number of cases where only that solver found a feasible solution (and the other none); `Oinf` is the similar value for infeasible cases; `Bpri` and `Bdua` are the numbers of instances for each solver when it has found a better primal/dual bound if both had one, respectively.

Gurobi outperforms Chuffed on over 100 instances, and even CBC outperforms Chuffed on over 50 instances. The differences between the MIP solvers and the Challenge solvers are particularly evident on proofs of optimality and feasibility (`Oopt` and `Ofea`). In this comparison, CBC outperforms Chuffed on 30 instances, while CPX only outperforms Chuffed on 8 instances.

What emerges from these tests is that, now that the linearization of CP models has been improved, MIP shows complementary strengths in contrast with the Challenge solvers. Given a new MiniZinc model it makes sense to try evaluating it with different classes of solvers, including MIP.

Table 2. Comparison of difference sets

Solver+config	Oopt	Ofea	Oinf	Bpri	Bdua
	Gurobi vs Chuffed				
Gurobi	52	22	0	41	-
Chuffed	49	45	2	23	-
	CPLEX(noDR) vs Chuffed				
CPLEX(noDR)	40	19	0	42	-
Chuffed	56	50	2	29	-
	CBC vs Chuffed				
CBC	21	9	0	25	-
Chuffed	92	112	3	24	-
	Gurobi best cfg vs CPLEX best cfg				
Gurobi	21	19	0	46	98
CPLEX(noDR)	3	14	0	33	31
	Chuffed vs Opturion CPX				
Chuffed	30	26	0	49	-
CPX	3	5	0	37	-

The work that has been done ensuring the behaviour of CBC is sound can also pay off by enabling constraint programmers to have immediate access to a free MIP solver adding an additional weapon to the CP modeller's armoury.

5   Conclusion

The results show that MIP solvers are highly competitive with CP solvers on MiniZinc benchmarks, which are, for the most part, written with CP solvers in mind. This is good news since it validates the constraint programming view of modelling: that the model should be written in the highest level possible, and it should be up to tools to map this to a suitable form for the solver if needed (of course the CP perspective is that the preferred mapping would be to a global propagator, but also MIP solvers might start providing global constraint handlers). This is also a challenge to CP since it illustrates that MIP solvers can be used out of the box to tackle problems that we often consider are suited to the CP solving technology. Of course there is a place for both CP and MIP solving technology, and one of the aims of a solver-independent modelling language is to avoid users committing early to the wrong technology. Better linearization makes MiniZinc a more attractive modelling language for the general OR community, which may then make them more aware of the CP view of modelling and solving.

There is plenty of scope for further improvement of automatic linearization of MiniZinc models. Issues that we plan to investigate are: better continuous

relaxations of nonlinear expressions, avoiding symmetry creation in decompositions, providing a declarative interface to domain refinement to allow the user to control the process using annotations.

References

1. A list of constraint languages (2016). http://www.csplib.org/Languages/
2. Andrei, N.: Introduction to GAMS Technology. Nonlinear Optimization Applications Using the GAMS Technology. Springer Optimization and Its Applications, vol. 81, pp. 9–23. Springer, New York (2013)
3. Beldiceanu, N., Carlsson, M., Demassey, S., Petit, T.: Global constraint catalogue: past, present and future. Constraints **12**(1), 21–62 (2007)
4. Björdal, G., Monette, J.-N., Flener, P., Pearson, J.: A constraint-based local search backend for MiniZinc. Constraints **20**(3), 325–345 (2015)
5. Chu, G.: Improving combinatorial optimization - extended abstract. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence IJCAI 2013, Beijing, China, pp. 3116–3120 (2013)
6. Chu, G.: Constraint Programming solver Chuffed (2016). https://github.com/geoffchu/chuffed. Accessed 16 Mar 2016
7. Cire, A.A., Hooker, J.N., Yunes, T.: Modeling with metaconstraints and semantic typing of variables. INFORMS JoC **28**(1), 1–13 (2016)
8. Côté, M.-C., Gendron, B., Rousseau, L.-M.: Modeling the regular constraint with integer programming. In: Van Hentenryck, P., Wolsey, L.A. (eds.) CPAIOR 2007. LNCS, vol. 4510, pp. 29–43. Springer, Heidelberg (2007)
9. Desrochers, M., Laporte, G.: Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. Oper. Res. Lett. **10**(1), 27–36 (1991)
10. Fourer, R., Gay, D.M.: Extending an algebraic modeling language to support constraint programming. INFORMS J. Comput. **14**(4), 322–344 (2002)
11. Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual Version 6.5. Houston, Texas: Gurobi Optimization (2016)
12. Hardin, J.R., Nemhauser, G.L., Savelsbergh, M.W.P.: Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements. Discrete Optim. **5**(1), 19–35 (2008)
13. Hooker, J.N.: Integrated Methods for Optimization. International Series in Operations Research & Management Science, vol. 170. Springer, US (2012)
14. IBM Software. IBM ILOG CPLEX optimizer. Data sheet, IBM Corporation (2014)
15. Koch, T.: Rapid mathematical prototyping. Ph.d. thesis, Technische Universität Berlin (2004)
16. Leo, K., Tack, G.: Multi-pass high-level presolving. In: International Joint Conference on Artificial Intelligence (IJCAI) (2015)
17. Linderoth, J., Ralphs, T.: Noncommercial software for mixed-integer linear programming. Technical report, Lehigh University (2004)
18. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I - convex underestimating problems. Math. Program. **10**(1), 147–175 (1976)
19. McKinnon, K., Williams, H.: Constructing integer programming models by the predicate calculus. Ann. Oper. Res. **21**, 227–245 (1989)
20. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. J. ACM **7**(4), 326–329 (1960)

21. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.R.: Mini-Zinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)
22. Opturion Pty Ltd. Opturion CPX user's guide: version 1.0.2 (2013)
23. Refalo, P.: Linear formulation of constraint programming models and hybrid solvers. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 369–383. Springer, Heidelberg (2000)
24. Salvagnin, D.: Detecting semantic groups in MIP models. In: Quimper, C.-G., Cavallo, M. (eds.) CPAIOR 2016. LNCS, vol. 9676, pp. 329–341. Springer, Heidelberg (2016). doi:10.1007/978-3-319-33954-2_24
25. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints **16**(3), 250–282 (2011)
26. Schutt, A., Stuckey, P.J., Verden, A.R.: Optimal carpet cutting. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 69–84. Springer, Heidelberg (2011)
27. Stuckey, P.J., Becket, R., Fischer, J.: Philosophy of the MiniZinc challenge. Constraints **15**(3), 307–316 (2010)
28. Stuckey, P.J., Tack, G.: MiniZinc with functions. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013. LNCS, vol. 7874, pp. 268–283. Springer, Heidelberg (2013)

Impact of SAT-Based Preprocessing
on Core-Guided MaxSAT Solving

Jeremias Berg[(✉)] and Matti Järvisalo

Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Helsinki, Finland
`jeremias.berg@cs.helsinki.fi`

Abstract. We present a formal analysis of the impact of Boolean satisfiability (SAT) based preprocessing techniques on core-guided solvers for the constraint optimization paradigm of maximum satisfiability (MaxSAT). We analyze the behavior of two solver abstractions of the core-guided approaches. We show that SAT-based preprocessing has no effect on the best-case number of iterations required by the solvers. This implies that, with respect to best-case performance, the potential benefits of applying SAT-based preprocessing in conjunction with core-guided MaxSAT solvers are in principle solely a result of speeding up the individual SAT solver calls made during MaxSAT search. We also show that, in contrast to best-case performance, SAT-based preprocessing can improve the worst-case performance of core-guided approaches to MaxSAT.

1 Introduction

Real-world applications [1–18] of maximum satisfiability (MaxSAT) [19–21], the optimization counterpart of the famous Boolean satisfiability problem (SAT) [22,23], are increasing in numbers as recent breakthroughs in MaxSAT solvers [24–32] are making MaxSAT more and more competitive as a constraint optimization paradigm.

A great majority of state-of-the-art MaxSAT solvers for solving optimization problems from the real world are *core-guided* [20,21], heavily relying on the power of SAT solvers as very effective means of proving unsatisfiability of subsets of soft constraints, or *unsat cores*, in an iterative fashion towards an optimal solution. Thus new breakthroughs in techniques for speeding up SAT solvers also have the potential of directly speeding up MaxSAT solvers further. One particularly fruitful line of research on speeding up SAT solvers has been the development of effective preprocessing techniques [33–35], applied most typically before search, as well as most recently also as inprocessing [34], i.e., during SAT search. Compared to SAT, preprocessing for MaxSAT has seen some but arguably less progress so far [26,30,36–39]. Recently, ways of employing preprocessing techniques developed for pure SAT in the context of MaxSAT have

been explored [26,30,40]. However, the impact of SAT-based preprocessing for MaxSAT solving seems to often be somewhat more modest than in the context of SAT solving [26,30,40]. The exact reasons for this difference are currently unclear; specifically, we are not aware of studies towards fundamental understanding on the potential of SAT-based preprocessing in the context of MaxSAT.

In this paper, we aim at providing further understanding on the potential of SAT-based preprocessing techniques in speeding up modern MaxSAT solvers. More specifically, we formally analyze the impact of SAT-based preprocessing techniques on the best-case and worst-case behavior of core-guided MaxSAT solvers [41–43]. As the basis of our analysis, we focus on two abstractions of MaxSAT solvers which together cover a number of modern core-guided MaxSAT solvers [25,30,42]. As the formal metric, we focus on the impact of SAT-based preprocessing on the best-case and worst-case number of iterations, which—although not the only possible metric—is a natural choice of metric applied in the literature for analyzing iterative SAT-based approaches in various problem settings [41–45] and which has also been subjected to some extent to empirical analysis for understanding specific MaxSAT solving approaches [46].

As the main contributions, considering *best-case performance* of the abstract core-guided solvers, we show that SAT-based preprocessing *has no effect* on the number of iterations required by the solvers. In fact, this is true regardless of assumptions on the type of cores (guaranteed-minimal or not) the underlying SAT solver (unsat core extractor) provides to the MaxSAT solvers; thus our analysis also sheds light on the impact of core minimization on the performance of the abstract core-guided solvers. Essentially, our results imply that, in terms of best-case performance—assuming optimal search heuristics—the potential benefits of applying SAT-based preprocessing in conjunction with core-guided MaxSAT solvers are solely a result of speeding up the individual SAT solver calls made during MaxSAT search. Furthermore, contrasting the results for best-case behavior, we also show that SAT-based preprocessing does, in cases, improve *worst-case* performance of core-guided MaxSAT solvers (without ever having a negative effect on the worst-case number of iterations).

This paper is organized as follows. After preliminaries on MaxSAT and SAT-based preprocessing for MaxSAT (Sect. 2), we detail abstractions of core-guided MaxSAT solvers we focus on (Sect. 3). Before detailed proofs of our results (provided in Sects. 5 and 6), we present a detailed overview of the main contributions (Sect. 4).

2   Preliminaries

Maximum satisfiability. For every Boolean variable x there are two literals: the positive literal x and the negative literal $\neg x$. A clause C is a disjunction of literals, and a CNF formula F is a conjunction of clauses. When convenient, we treat a clause as a set of literals and a CNF formula as a set of clauses. We denote by $\text{VAR}(F)$ the set of variables appearing in F. A truth assignment is a function $\tau \colon \text{VAR}(F) \to \{0, 1\}$. A clause C is satisfied by τ if $\tau(l) = 1$ for a

positive literal or $\tau(l) = 0$ for a negative literal $l \in C$. A CNF formula F is satisfied by τ if τ satisfies all clauses $C \in F$. A formula F is satisfiable if there is a truth assignment that satisfies it, otherwise it is unsatisfiable.

A (weighted partial) MaxSAT instance $F = (F_h, F_s, w)$ consists of two CNF formulas, F_h (hard clauses) and F_s (soft clauses), together with a function $w \colon F_s \to \mathbb{N}$ assigning a positive weight $w(C)$ to each $C \in F_s$. If $w(C) = 1$ for all $C \in F_s$, the instance is unweighted. An (unsatisfiable) *core* of a MaxSAT instance F is a subset $\kappa \subseteq F_s$ such that $\kappa \wedge F_h$ is unsatisfiable. A core is minimal (a MUS) if no $\kappa_s \subset \kappa$ is a core of F. We denote the set of all MUSes of F by $\mathsf{mus}(F)$. For a subset $S \subseteq F_s$ and clause $C \in S$, C is *necessary* for S if $F_h \wedge S$ is unsatisfiable and $F_h \wedge (S \setminus \{C\})$ is satisfiable.

An assignment τ that satisfies F_h is a *solution* to a MaxSAT instance F. For a solution τ, let $\mathrm{COST}(F, \tau) = \sum_{C \in F_s} w(C) \cdot (1 - \tau(C))$, i.e., the sum of the weights of soft clauses in F not satisfied by τ. A solution τ is optimal if $\mathrm{COST}(F, \tau) \leq \mathrm{COST}(F, \tau')$ for every solution τ'; we denote the cost of F, i.e., the value $\mathrm{COST}(F, \tau)$ for optimal solutions τ, by $\mathrm{COST}(F)$. Given a MaxSAT instance F, the MaxSAT problem asks to find an optimal solution to F.

SAT-Based Preprocessing for MaxSAT. Preprocessing is today an integral part of SAT solving [33,34]. Consisting of applying a combination of satisfiability-preserving simplification (or rewriting) rules on the input CNF formula F to obtain a preprocessed CNF formula $\mathsf{pre}(F)$, a central aim of preprocessing is to speed up the runtime of a SAT solver so that the combined preprocessing time and solving time on $\mathsf{pre}(F)$ is shorter than the runtime of the solver on F. Several preprocessing techniques for SAT have been proposed. In this work we will focus on bounded variable elimination, subsumption elimination, self-subsuming resolution, and blocked clause elimination, as perhaps the most common preprocessing techniques in modern SAT solving.

Resolution. Given two clauses $C = C_1 \vee l$ and $D = D_1 \vee \neg l$ of F, the *resolution rule* states that the clause $C \bowtie_l D = C_1 \vee D_1$, called the *resolvent*, can be inferred by *resolving* on the literal l. This is lifted to two sets $S_l \subseteq F$ and $S_{\neg l} \subseteq F$ of clauses that contain the literal l and $\neg l$, respectively, by $S_l \bowtie_l S_{\neg l} = \{C \bowtie_l D \mid C \in S_l, D \in S_{\neg l}, \text{ and } C \bowtie_l D \text{ is not a tautology}\}$.

Bounded Variable Elimination (BVE) [33]. For a variable $x \in \mathrm{VAR}(F)$, denote by F_x ($F_{\neg x}$) the clauses of F containing the literal x ($\neg x$). If $|F_x \bowtie_x F_{\neg x}| \leq |F_x \cup F_{\neg x}|$, the BVE rule allows converting the formula F to $(F \setminus (F_x \cup F_{\neg x})) \cup (F_x \bowtie_x F_{\neg x})$.

Subsumption Elimination (SE). A clause $C \in F$ subsumes another clause $D \in F$ if $C \subseteq D$. The SE rule allows for removing subsumed clauses from F.

Self-Subsuming Resolution (SSR). Given two clauses $C, D \in F$ s.t. $C = C_1 \vee l$, $D = D_1 \vee \neg l$ for a literal l and $D_1 \subseteq C_1$, the SSR rule allows for replacing C by C_1.

Blocked Clause Elimination (BCE) [47]. A clause $C \in F$ is blocked if it contains a literal $l \in C$ s.t $C \bowtie_l D$ is a tautology for all $D \in F_{\neg l}$. BCE allows removing blocked clauses from F.

Example 1. Consider the CNF formula
$F = \{(x \vee y), (\neg t \vee \neg z), (\neg z \vee y), (\neg y \vee z), (z \vee t), (x), (y \vee t), (z \vee t \vee x)\}$. Due to the clause (x), SE allows for removing $(x \vee y)$ and $(z \vee t \vee x)$. After this, using BVE to eliminate z, results in the formula $\mathsf{pre}(F) = \{(\neg t \vee \neg y), (t \vee y), (x)\}$.

As shown in [26], many important SAT preprocessing techniques, including BVE, SE, and SSR, cannot be used directly on MaxSAT instances. However, a correct lifting on these techniques for MaxSAT is enabled by the so-called *labelled CNF* (LCNF) framework [26,48]. The LCNF framework enables correct applications of SAT-based preprocessing techniques on a MaxSAT instance $F = (F_h, F_s, w)$ using the procedure outlined in Fig. 1. Each soft clause $C \in F_s$ is augmented with a fresh *label variable* l_C (Step 1). Then SAT preprocessing is applied on the CNF formula $F_h \cup F_s^a$ (Step 2). To ensure correctness in terms of MaxSAT, the preprocessor needs to be restricted from resolving on any of the label variables. The hard clauses of $\mathsf{pre}(F)$ are the clauses output by the SAT preprocessor on $F_h \cup F_s^a$ (Step 3). The soft clauses of $\mathsf{pre}(F)$ contain a unit negation of each label variable that has not been eliminated by preprocessing; the weight function w^P assigns to each $(\neg l_C)$ the same weight as was assigned to C by w (Step 4). Finally, the procedure returns the preprocessed instance $\mathsf{pre}(F) = (\mathsf{pre}(F)_h, \mathsf{pre}(F)_s, w^P)$ (Step 5). The soft clauses of $\mathsf{pre}(F)$ are all unit soft clauses $(\neg l_C)$ where the variable l_C was added to some soft clause $C \in F_s$ of the original instance F in Step 1. Due to BVE, the variable l_C might appear in more than one hard clause of $\mathsf{pre}(F)$ and there might be literals that have been eliminated entirely from the formula during preprocessing.

1. $F_s^a = \{(C \vee l_C) \mid C \in F_s, \; l_C \text{ is a fresh variable}\}$.
2. Run VE, SSR, SE, and BCE on $F_h \cup F_s^a$ until fixpoint to obtain $\mathsf{pre}\,(F)_h$.
3. $\mathsf{pre}\,(F)_s = \{(\neg l_C) \mid \exists C' \in \mathsf{pre}\,(F)_h, l_C \in C'\}$.
4. $w^P((\neg l_C)) = w(C)$ for all $(\neg l_C) \in \mathsf{pre}\,(F)_s$.
5. Return $\mathsf{pre}\,(F) = (\mathsf{pre}\,(F)_h, \mathsf{pre}\,(F)_s, w^P)$.

Fig. 1. Applying SAT-based preprocessing to MaxSAT instance $F = (F_h, F_s, w)$.

Example 2. Let $F = (F_h, F_s)$ be an unweighted MaxSAT instance with $F_h = \{(x \vee y), (z), (z \vee t)\}$ and $F_s = \{(\neg x), (\neg y), (\neg t)\}$. Augmenting the soft clauses with the label variables l_1, l_2, and l_3 to form $F_s^a = \{(\neg x \vee l_1), (\neg y \vee l_2), (\neg t \vee l_3)\}$, and applying SAT-based preprocessing (BVE and SE) results in the instance $\mathsf{pre}(F)$ with $\mathsf{pre}(F)_h = \{(l_1 \vee l_2), (z)\}$ and $\mathsf{pre}(F)_s = \{(\neg l_1), (\neg l_2)\}$. Notice that preprocessing eliminates the label l_3.

Correctness of SAT-based preprocessing for MaxSAT is summarized as follows [26].

Theorem 1 ([26]). *Let F be a MaxSAT instance and $\mathsf{pre}(F)$ the instance resulting from preprocessing F according to the procedure in Fig. 1. The following*

hold: (i) $\text{COST}(F) = \text{COST}(\textit{pre}(F))$; (ii) any optimal solution to $\textit{pre}(F)$ restricted to $\text{VAR}(F)$ is an optimal solution to F; and (iii) $\{C_1, \ldots, C_n\} \in \textit{mus}(F)$ iff $\{(\neg l_{C_1}), \ldots, (\neg l_{C_n})\} \in \textit{mus}(\textit{pre}(F))$.

3 Core-Guided MaxSAT Algorithms

In this section we detail the two abstractions of MaxSAT algorithms we analyze in this work: *CG* and *HS*. Both are examples of so-called *core-guided MaxSAT solvers*, one of the most successful current MaxSAT solving approaches with several variants, e.g. [28,31,42,49–52]. *CG* (Fig. 2 left) is the same abstraction as studied in [53]. CG works by iteratively calling a SAT solver to extract unsatisfiable cores and ruling out each of the found cores by exploiting *cardinality constraints*. HS (Fig. 2 right) follows the implicit hitting set approach to MaxSAT [54,55], iteratively using a SAT solver to extract unsatisfiable cores, and an exact minimum-cost hitting set algorithm to compute hitting sets over the found cores.

In more detail, at each iteration i, CG checks the satisfiability of a working formula F_w^i, which initially contains all clauses in the input formula, using a SAT solver. If F_w^i is satisfiable, CG returns the satisfying assignment τ returned by the SAT solver restricted onto the variables of F. Otherwise, the SAT solver returns a core κ^i of F_w^i. Finally, CG forms the next working formula F_w^{i+1} by processing the core κ^i. The exact method in which CG processes κ^i is left abstract. Following [53], we consider algorithms that extend soft clauses with blocking variables and impose hard linear (in)equalities over the blocking variables. More precisely, CG is allowed to modify the soft clauses $C \in F_s^i$ by two operations: **Relax**(C) and **Clone**(C, w).

CG:
$F_w^1 \leftarrow F_h \cup F_s$
for *i=1... * **do**
    $(result, \kappa, \tau) \leftarrow \text{SATSOLVE}(F_w^i)$
    if *result="satisfiable"* **then**
        | return τ // optimal solution
    else
        // SAT solver returned unsat core
        $F_w^i = (F_w^i \setminus \kappa)$
        $F_w^{i+1} \leftarrow \text{PROCESS}(F_w^i, \kappa)$
    end
end

HS:
$\mathcal{K} \leftarrow \emptyset$ // set of found unsat cores of F
$F_w \leftarrow (F_h \cup F_s)$
while *true* **do**
    $H \leftarrow \text{MINCOSTHITTINGSET}(\mathcal{K})$
    $F_w \leftarrow F_h \cup (F_s \setminus H)$
    $(result, \kappa, \tau) \leftarrow \text{SATSOLVE}(F_w)$
    if *result="satisfiable"* **then**
        | return τ // optimal solution
    else
        // SAT solver returned unsat core
        | $\mathcal{K} \leftarrow \mathcal{K} \cup \{\kappa\}$
    end
end

Fig. 2. Abstractions of MaxSAT solvers: CG (left) and HS (right), given a MaxSAT instance $F = (F_h, F_s, w)$ as input.

– **Relax**(C) allows replacing C by $C \vee b$ where b is a new *blocking variable* not appearing anywhere else in the formula.
– **Clone**(C, w) allows adding a soft duplicate C' of C to the formula and relaxing C' by calling **Relax**(C'). The (relaxed) *clone* C' is assigned weight w, and w is subtracted from the weight of C (C is discarded once it has weight 0).

In addition to these operations, CG is also allowed to add hard linear (in)equalities (cardinality, or more precisely, pseudo-Boolean, constraints) over the blocking variables. Given a cardinality constraint $\sum w_i \cdot x_i \circ K$ over variables x_i, constants w_i, and $\circ \in \{=, <, \leq\}$, we denote by $\mathrm{CNF}(\sum w_i \cdot x_i \circ K)$ a CNF encoding of such a constraint. Following most core-guided MaxSAT algorithm implementations, we place two important restrictions on how CG can process the cores it encounters. First, the cardinality constraints are not allowed to mention any of the variables in the initial formula F. Second, if the algorithm extracts n cores during solving an instance F, and w_m^i is the smallest weight over all clauses in the ith core extracted, the optimum cost of F is $\mathrm{COST}(F) = \sum_{i=1}^{n} w_m^i$. A concrete example of an algorithm fitting the CG model is the WPM1 algorithm [50], concurrently proposed as WMSU1 [51], as an extension of the classical Fu-Malik algorithm [49] to weighted MaxSAT. Given a core κ^i, WPM1 first computes w_m^i. Then it calls **Clone**(C^i, w_m^i) for each $C^i \in \kappa^i$ and adds an *exactly-one* constraint over the blocking variables added during the cloning operation.

HS is a hybrid algorithm, instantiated in [25,55], that uses a SAT solver for core extraction from a working formula F_w, initially all clauses of the working formula. Given a collection $\mathcal{K}$ of extracted cores, HS uses an exact algorithm (an integer programming solver in practice) to find a minimum-cost hitting set hs over $\mathcal{K}$. The working formula is then updated to contain all clauses of F except for the soft clauses in hs, and the SAT solver invoked again. If the working formula is satisfiable, the satisfying assignment obtained is an optimal solution to F. Otherwise another core is obtained and the search continues with hitting set computation.

4   Overview of Results

In this section we give an overview of the main contributions of this paper. The algorithm-dependent formal proofs are provided after this overview in Sects. 5 and 6.

We start by first defining the metric with respect to which we perform the formal analysis. The definition, intuitively matching with the number of iterations made by the abstract MaxSAT solvers considered, relies on the concept of *core traces*. Informally, a core trace T is a finite sequence of MaxSAT cores matching a possible execution of a core-guided MaxSAT solver. More formally, given a MaxSAT instance F and $\mathcal{A} \in \{\mathrm{CG}, \mathrm{HS}\}$, a sequence $(\kappa^1, \ldots, \kappa^n)$ of cores is an $\mathcal{A}$ *core trace* on F if there exists an execution of $\mathcal{A}$ on F such that (i) the core extracted by $\mathcal{A}$ at iteration i is κ^i; and (ii) $\mathcal{A}$ terminates after having encountered all cores in the sequence (i.e., the $(n+1)$th SAT solver call is satisfiable). For a core

trace T, we denote by $|T|$ the number of cores in T, i.e., the *length* of T. Whenever appropriate, we refer to $\mathcal{A}$ core traces on F simply as $\mathcal{A}$ traces on F.

As the metric under analysis, we consider both the *minimum* and *maximum length* over all possible $\mathcal{A}$ traces for different choices of $\mathcal{A}$. More specifically, for $\mathcal{A} \in \{\text{CG}, \text{HS}\}$, we analyze the relative minimum and maximum lengths of core traces for the following variants of $\mathcal{A}$.

- $\mathcal{A}_{\text{pre}}$: $\mathcal{A}$ applied after SAT-based preprocessing (recall Fig. 1).
- $\mathcal{A}^{\text{mus}}$: $\mathcal{A}$ using a SAT solver that is guaranteed to return a MUS when invoked on an unsatisfiable formula (notice that an $\mathcal{A}^{\text{mus}}$ trace contains only MUSes).
- $\mathcal{A}^{\text{mus}}_{\text{pre}}$: $\mathcal{A}^{\text{mus}}$ applied after SAT-based preprocessing.

For a MaxSAT instance F, we denote by $\mathsf{minlen}(\mathcal{A}, F)$ and $\mathsf{maxlen}(\mathcal{A}, F)$ the minimum and maximum length $\mathcal{A}$ traces on F, respectively, or in other words, the best-case and worst-case number of iterations required by $\mathcal{A}$ for solving F.
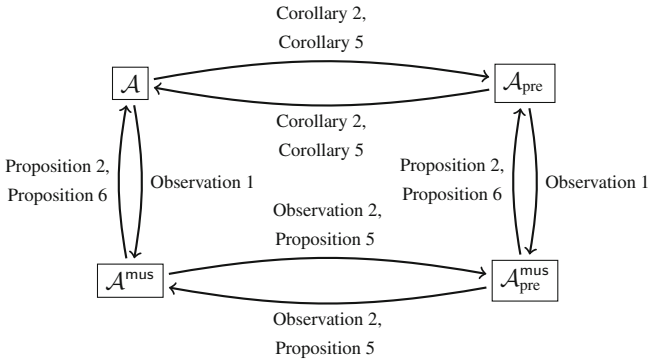
Fig. 3. Best-case performance in the number of iterations of $\mathcal{A} \in \{\text{CG}, \text{HS}\}$. Here $X \to Y$ iff $\mathsf{minlen}(X, F) \leq \mathsf{minlen}(Y, F)$ for all instances F.

Results. We provide a full characterization of the effect of preprocessing on the maximum and minimum length of core traces on F. The results on the best-case performance (minimum lengths of core traces) are summarized in Fig. 3 for $\mathcal{A} \in \{\text{CG}, \text{HS}\}$. In the figure, an edge $X \to Y$ indicates that, for any MaxSAT instance F, the shortest X core trace on F is at most as long as the shortest Y core trace on F. Analogously, our results for the worst-case performance (maximum lengths of core traces) are summarized in Fig. 4. Here the edge $X \to Y$ indicates that, for any MaxSAT instance F, the longest X core trace on F is at most as long as the longest Y core trace on F; $X \nrightarrow Y$ indicates that $X \to Y$ does not hold. In words, we will provide in the following sections detailed proofs for the fact that SAT-based preprocessing cannot lower the minimum number of iterations required by CG or HS. For some intuition, we will show that for $\mathcal{A} \in \{\text{CG}, \text{HS}\}$, one of the shortest $\mathcal{A}$ core traces on any MaxSAT instance F is also a $\mathcal{A}^{\text{mus}}$ trace, and that preprocessing cannot alter the MUS structure nor the $\mathcal{A}^{\text{mus}}$ traces on F.
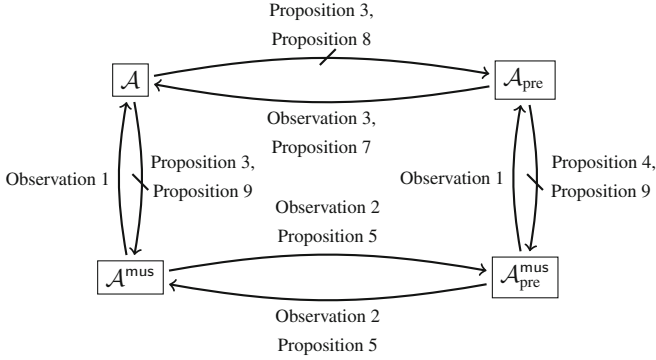
Fig. 4. Worst-case performance in the number of iterations of $\mathcal{A} \in \{\mathrm{CG}, \mathrm{HS}\}$. Here $X \to Y$ iff $\mathsf{maxlen}(X, F) \leq \mathsf{maxlen}(Y, F)$ for all F, and $X \nrightarrow Y$ indicates that $X \to Y$ does not hold.

In contrast, we will also show that preprocessing can improve the worst-case performance of both of the algorithms. Intuitively, this is due to the fact that preprocessing can remove soft clauses that are not members of any MUSes of F and hence do not contribute to the unsatisfiability of F, but still might force either algorithm to iterate unnecessarily many times.

   We proceed now throughout Sects. 5 and 6 by providing formal proofs for all of the results summarized in Figs. 3 and 4. Before the more involved proofs, we start with an algorithm-independent observation and an auxiliary result that makes the remaining proofs simpler by allowing us to assume MaxSAT instances to have a specific form without loss of generality.

Observation 1. *For $\mathcal{A} \in \{CG, HS\}$ and any MaxSAT instance F, any $\mathcal{A}^{\mathsf{mus}}$ trace on F is also an $\mathcal{A}$ trace on F. Hence $\mathsf{maxlen}(\mathcal{A}^{\mathsf{mus}}, F) \leq \mathsf{maxlen}(\mathcal{A}, F)$ and $\mathsf{minlen}(\mathcal{A}^{\mathsf{mus}}, F) \geq \mathsf{minlen}(\mathcal{A}, F)$.*

   Finally, in the remaining proofs, we will use the fact that Theorem 1 guarantees that SAT-based preprocessing does not affect the set of MUSes of F in terms of of the mapping $(\neg l_C) \to C$ between the soft clauses of $\mathsf{pre}(F)$ and F. In order to avoid explicitly referring to this mapping in every proof, we will employ a technical observation from [40]. More specifically, *we will assume for the remaining part of this paper* that the soft clauses $C \in F_s$ of each MaxSAT instance F have already been augmented with label variables l_C to form the hard clause $C \vee l_C$ and the soft clause $(\neg l_C)$. In other words, we will assume that all soft clauses of F are unit negative literals $(\neg l_C)$ with the variable l_C not appearing negatively in any other clause and only appearing positively among the hard clauses. Under this assumption, the literals appearing in the soft clauses of F can be reused as label variables while preprocessing [40], thus removing the need of adding any new variables. Hence $\mathsf{pre}(F)_s \subseteq F_s$, and Theorem 1 can be simplified.

Corollary 1 (of Theorem 1). *Let F be a MaxSAT instance and $\mathsf{pre}(F)$ the instance resulting after preprocessing F. Then $\mathsf{mus}(F) = \mathsf{mus}(\mathsf{pre}(F))$.*

Most importantly, our assumption on the form of MaxSAT instances *does not affect core traces*. A proof for this auxiliary result is provided in Appendix A.

Proposition 1. *Let $F = (F_h, F_s, w)$ be a MaxSAT instance, and $F^P = (F_h \cup F_s^a,$ $F_s^P, w^P)$ the MaxSAT instance with $F_s^a = \{C \vee l_C \mid C \in F_s,\ l_C$ is a fresh variable$\}$, $F_s^P = \{(\neg l_C) \mid C \in F_s\}$, and $w^P((\neg l_C)) = w(C)$. The following observations hold.*

1. $\mathrm{COST}(F) = \mathrm{COST}(F^P)$, and the optimal solutions of F are the same as the optimal solutions of F^P restricted to $\mathrm{VAR}(F)$.
2. For $\mathcal{A} \in \{HS, CG\}$, there is a one-to-one mapping between the $\mathcal{A}$ core traces on F and F^P of equal length.

5   Impact of Preprocessing on HS

We continue with formal proofs of our main results for HS. An essential intuition for these proofs is that HS only extracts cores of the original instance. In other words, an HS core trace on any F only contains cores of the original instance F.

We first analyze best-case performance. The first observation shows that preprocessing does not affect the lengths of HS MUS traces in a significant way.

Observation 2.
For any MaxSAT instance F, $\mathsf{minlen}(HS^{\mathsf{mus}}, F) = \mathsf{minlen}(HS_{\mathsf{pre}}^{\mathsf{mus}}, F)$.

Proof. (Sketch) By Corollary 1 we obtain $\kappa \in \mathsf{mus}(F)$ iff $\kappa \in \mathsf{mus}(\mathsf{pre}(F))$. The fact that an HS^{mus} trace on F only contains MUSes of F implies that T is an HS^{mus} trace on F iff it is an $HS_{\mathsf{pre}}^{\mathsf{mus}}$ trace on F. $\qquad\square$

Next we show that executions of HS^{mus} are always shortest executions of HS.

Proposition 2.
For any MaxSAT instance F, $\mathsf{minlen}(HS, F) \geq \mathsf{minlen}(HS^{\mathsf{mus}}, F)$ and $\mathsf{minlen}(HS_{\mathsf{pre}}, F) \geq \mathsf{minlen}(HS_{\mathsf{pre}}^{\mathsf{mus}}, F)$.

Proof. We will show that $\mathsf{minlen}(\mathrm{HS}, F) \geq \mathsf{minlen}(\mathrm{HS}^{\mathsf{mus}}, F)$ for any F, and thus $\mathsf{minlen}(\mathrm{HS}_{\mathsf{pre}}, F) \geq \mathsf{minlen}(\mathrm{HS}_{\mathsf{pre}}^{\mathsf{mus}}, F)$ as well. Let $T = (\kappa^1, \ldots, \kappa^n)$ be an arbitrary HS core trace on F. Let hs^* be a minimum-cost hitting set over $\{\kappa^1, \ldots, \kappa^n\}$ for which $F \setminus hs^*$ is satisfiable. The statement follows by constructing an $\mathrm{HS}^{\mathsf{mus}}$ trace T_m on F s.t. $|T_m| \leq |T|$. As each $\kappa^i \in T$ is a core of F, all contain at least one MUS $m \subseteq \kappa^i$. Consider the set $\mathcal{M}$ of at most n MUSes of F constructed as follows. (1) Let $\mathcal{M}^1 = \{m^1\}$, where m^1 is any MUS contained in κ^1; (2) let $\mathcal{M}^i = \mathcal{M}^{i-1} \cup \{m^i\}$, where $m^i \subseteq \kappa^i$ is a MUS such that $m^i \notin \mathcal{M}^{i-1}$ if any exist, else let $\mathcal{M}^i = \mathcal{M}^{i-1}$. We obtain $\mathcal{M}^n = \mathcal{M}$ of size $|\mathcal{M}| = k \leq n$ such that each $m \in \mathcal{M}$ is a subset of some $\kappa^i \in T$.

We show that $\mathcal{M}$ can be ordered to form an $\mathsf{HS}^{\mathsf{mus}}$ trace on F of length at most k, since if a minimum-cost hitting set hs over any proper subset $\mathcal{M}_s \subset \mathcal{M}$ hits all $m \in \mathcal{M}$, then hs^* is also a minimum-cost hitting set over $\mathcal{M}_s$, and $\mathsf{HS}^{\mathsf{mus}}$ can terminate. As $F \setminus hs^*$ is satisfiable, hs^* is also a hitting set over $\mathcal{M}$ and over $\mathcal{M}_s$. Furthermore, as each $m \in \mathcal{M}$ is a subset of some $\kappa^i \in T$ and each $\kappa^i \in T$ contains a MUS in $\mathcal{M}$, hs^* is a minimum-cost hitting set of $\mathcal{M}$. Finally, as hs is a hitting set over $\mathcal{M}$ the cost of hs is not less than the cost of hs^*. Hence hs^* is a minimum-cost hitting set of $\mathcal{M}_s$, so the hitting set computation could have returned hs^*, thus allowing $\mathsf{HS}^{\mathsf{mus}}$ to terminate. $\qquad\square$

A simple corollary is that shortest executions of HS and $\mathsf{HS}_{\mathsf{pre}}$ are of equal length.

Corollary 2. *For any MaxSAT instance F, $\mathsf{minlen}(HS, F) = \mathsf{minlen}(HS_{pre}, F)$.*

Proof.
Observation 1 and Proposition 2 establish $\mathsf{minlen}(\mathrm{HS}, F) = \mathsf{minlen}(\mathrm{HS}^{\mathsf{mus}}, F)$ and $\mathsf{minlen}(\mathrm{HS}_{\mathsf{pre}}, F) = \mathsf{minlen}(\mathrm{HS}^{\mathsf{mus}}_{\mathsf{pre}}, F)$. Together with Observation 2 this implies $\mathsf{minlen}(\mathrm{HS}, F) = \mathsf{minlen}(\mathrm{HS}^{\mathsf{mus}}, F) = \mathsf{minlen}(\mathrm{HS}^{\mathsf{mus}}_{\mathsf{pre}}, F) = \mathsf{minlen}(\mathrm{HS}_{\mathsf{pre}}, F)$. $\qquad\square$

We move on to the worst-case results. Corollary 1 can be used to show that valid executions of $\mathsf{HS}_{\mathsf{pre}}$ are also valid executions of HS on any MaxSAT instance.

Observation 3.
For any MaxSAT instance F, $\mathsf{maxlen}(HS_{pre}, F) \leq \mathsf{maxlen}(HS, F)$.

Proof. As $\mathsf{pre}(F)_s \subseteq F_s$ and any MUS of $\mathsf{pre}(F)$ is a MUS of F, any core of $\mathsf{pre}(F)$ is a core of F. $\qquad\square$

Finally for this section, we prove the three $X \nrightarrow Y$ edges in Fig. 4 for HS. For this, we need as a witness a family of MaxSAT instances $F(n)$ and a X core trace T on $F(n)$ s.t. $|T| > \mathsf{maxlen}(Y, F(n))$.

Proposition 3.
There is a family of MaxSAT instances $F(n)$ with $\mathcal{O}(n)$ soft clauses s.t. $\mathsf{maxlen}(HS, F(n)) \geq n$ and $\mathsf{maxlen}(HS^{mus}, F(n)) = \mathsf{maxlen}(HS_{pre}, F(n)) = 1$.

Proof. Fix n and let $F(n)_h = \{(x \vee y)\} \cup \{(x \vee y \vee z_i) \mid i = 1, \ldots, n\}$ and $F(n)_s = \{(\neg x), (\neg y)\} \cup \{(\neg z_i) \mid i = 1, \ldots, n\}$ with $w((\neg x)) = w((\neg y)) = n$ and $w((\neg z_i)) = 1$ for all i. Now $\mathrm{COST}(F(n)) = n$ and $\mathsf{mus}(F(n)) = \{\{(\neg x), (\neg y)\}\}$, explaining why $\mathsf{maxlen}(\mathrm{HS}^{\mathsf{mus}}, F(n)) = 1$. A linear-length HS core trace on $F(n)$ is $(\kappa^1, \ldots, \kappa^n)$, where $\kappa^i = \{(\neg x), (\neg y), (\neg z_i)\}$. HS cannot terminate before extracting all n cores. To see this, consider an earlier iteration $i < n$. The weight of the hitting set $\{(\neg z_j) \mid j = 1, \ldots, i\}$ over $\mathcal{K}^i = \{\kappa^1, \ldots, \kappa^i\}$ is $i < n = w((\neg x)) = w((\neg y))$ and as such any minimum-cost hitting set over $\mathcal{K}^i$ can not contain $(\neg x)$ or $(\neg y)$, preventing HS from terminating. Hence $\mathsf{maxlen}(\mathrm{HS}, F(n)) \geq n$.

However, due to the clause $(x \vee y)$, SE allows the removal of the clause $(x \vee y \vee z_i)$ for all i. Hence $\mathsf{pre}(F(n))$ has $\mathsf{pre}(F(n))_h = \{(x \vee y)\}$ and $\mathsf{pre}(F(n))_s = \{(\neg x), (\neg y)\}$. The only core of $\mathsf{pre}(F(n))$ is $\{(\neg x), (\neg y)\}$, and thus $\mathsf{maxlen}(\mathsf{HS_{pre}}, F(n)) = 1$.   □

Proposition 4. *For any n, there is a family of MaxSAT instances $F(n)$ with $\mathcal{O}(n)$ soft clauses s.t. $\mathsf{maxlen}(\mathsf{HS_{pre}}, F(n)) \geq n$ and $\mathsf{maxlen}(\mathsf{HS_{pre}^{mus}}, F) = 1$.*

Proof. Fix n and let

$$F(n)_h = \{(x_{1,2} \vee x_{1,3} \vee \neg x_{2,3}), (E \vee x_{2,3})\} \cup \tag{1}$$

$$\bigcup_{i=4}^{n+3} \{(x_{1,2} \vee x_{2,i} \vee \neg x_{1,i}), (x_{1,i} \vee x_{1,3} \vee \neg x_{3,i}), (x_{3,i} \vee x_{2,i} \vee \neg x_{2,3})\} \cup \tag{2}$$

$$\{(x_{T,x} \vee x_{x,y} \vee \neg x_{T,y}), (x_{T,x} \vee x_{T,y} \vee \neg x_{x,y}) \mid 1 \leq x, y \leq n+3\} \tag{3}$$

and $F(n)_s = \{(\neg x_{1,2}), (\neg x_{1,3}), (\neg E)\} \cup \{(\neg x_{2,i}) \mid i = 4, \ldots, n+3\}$ with $w((\neg x_{1,2})) = w((\neg x_{1,3})) = w((\neg E)) = n$ and $w((\neg x_{2,i})) = 1$ for all i. The hard clauses on row 3 are included in order to prevent preprocessing from simplifying $F(n)$ in any way. Intuitively, $F(n)$ encodes hard transitivity constraints over an undirected graph with each node having degree at least 4. Hence $\mathsf{pre}(F(n)) = F(n)$ at it suffices to show $\mathsf{maxlen}(\mathsf{HS}, F(n)) \geq n$ and $\mathsf{maxlen}(\mathsf{HS^{mus}}, F) = 1$. Both arguments are similar to Proposition 3. As $\mathsf{mus}(F(n)) = \{\{(\neg x_{1,2}), (\neg x_{1,3}), (\neg E)\}\}$, it follows that $\mathsf{maxlen}(\mathsf{HS^{mus}}, F) = 1$. A linear-length HS core trace on $F(n)$ is $(\kappa^1, \ldots, \kappa^n)$, where $\kappa^i = \{(\neg x_{1,2}), (\neg x_{1,3}), (\neg E), (\neg x_{2,i+3})\}$.   □

6   Impact of Preprocessing on CG

We start the analysis for CG by linking CG core traces with optimum cost.

Observation 4. *Let $T = (\kappa^1, \ldots, \kappa^n)$ be a CG or $\mathsf{CG^{mus}}$ core trace on a MaxSAT instance F, and $w^i = \min\{w(C^i) \mid C^i \in \kappa^i\}$. The cost of F is $\mathrm{COST}(F) = \sum_{i=1}^{n} w^i$.*

An important corollary of Observation 4 is that no proper subsequence of a CG or $\mathsf{CG^{mus}}$ core trace on F can in itself be a CG or $\mathsf{CG^{mus}}$ trace on F.

The proofs on CG, in contrast to HS, need to consider the fact that the ith core κ^i in a CG core trace on F is not a core of F, but rather, of the working formula F^i instead. Following this, a relationship between the cores of F^i and the cores of F was derived in [53]. After necessary definitions and restatement of the result of [53], we will prove an analogous result regarding the relationship between the MUSes of F^i and F, which proves useful for obtaining our main results for CG.

6.1   Cores and MUSes of Working Formulas of CG

We follow here definitions from [53]. Let F be a MaxSAT instance and F^i the working formula of CG on iteration i when invoked on F. Let $\mathbf{card}^i$ be the set of all cardinality constraints added to F by CG during iterations $1, \ldots, i$. Thus the hard clauses of F^i are $F^i_h = F_h \cup \mathbf{card}^i$. We denote by $\mathbf{soln}(\mathbf{card}^i)$ the set of truth assignments satisfying $\mathbf{card}^i$ and not assigning any of the variables in F. Given any $\tau \colon \mathrm{VAR}(F) \to \{0,1\}$ and $\alpha \in \mathbf{soln}(\mathbf{card}^i)$, $(\tau{:}\alpha)$ is the truth assignment over the variables of F^i that assigns all variables of F according to τ and the rest according to α; $(\tau{:}\alpha)$ is well-defined as the auxiliary cardinality constraints are not allowed to mention variables in F. For any $\beta \in \mathbf{soln}(\mathbf{card}^i)$ and $S^i \subseteq F^i_s$, the *reduction* of S^i wrt β, $S^i|_\beta$ is obtained by (1) removing from S^i all clauses satisfied by β; (2) removing from each remaining clause $C^i \in S^i$ all blocking variables, i.e., all literals falsified by β; and (3) setting the weights of each $C^i \in S^i$ back to their original weights in F (removing duplicates). The restriction $\mathrm{R}(C^i) \in F_s$ of a soft clause $C^i \in F^i_s$ is obtained by (1) removing all added blocking variables from C^i; (2) removing all clones of C^i from the instance; and (3) setting the weight of C^i back to its original weight in F. Restriction is lifted to a set $S^i \subseteq F^i_s$ by $\mathrm{R}(S^i) = \{\mathrm{R}(C^i) \mid C^i \in S^i\}$. Notice that $S^i|_\beta \subseteq \mathrm{R}(S^i) \subseteq F_s$. With these definitions we can now restate a central result from [53].

Theorem 2 (Adapted from [53]).
A set $\kappa^i \subseteq F^i_s$ is a core of F^i iff $\kappa^i|_\beta$ is a core of F for all $\beta \in \mathbf{soln}(\mathbf{card}^i)$.

We will now prove an analogous characterization of the MUSes of F^i.

Theorem 3. *A set $M^i \subseteq F^i_s$ is a MUS of F^i iff there is a collection $\Upsilon \subseteq \mathbf{mus}(F)$ s.t.*

1. $\mathrm{R}(M^i) = \bigcup_{M \in \Upsilon} M$;
2. *for each $M \in \Upsilon$, there is an $\alpha \in \mathbf{soln}(\mathbf{card}^i)$ s.t. $M \subseteq M^i|_\alpha$ and $M' \not\subseteq M^i|_\alpha$ for all other $M' \in \Upsilon$; and*
3. *for each $\alpha \in \mathbf{soln}(\mathbf{card}^i)$, there is an $M \in \Upsilon$ s.t. $M \subseteq M^i|_\alpha$.*

Note that condition 3 is equivalent to the requirement of Theorem 2 for the set M^i being a core of F^i, since $M^i|_\alpha \subseteq \mathrm{R}(M^i)$ and $M^i|_\alpha$ should be unsatisfiable for all α.

Before proving Theorem 3, consider the following example for more intuition.

Example 3. Consider the unweighted MaxSAT instance $F = (F_h, F_s)$ with $F_h = \{(x_1 \vee x_2 \vee x_3), (x_3 \vee x_4 \vee x_5), (x_5 \vee x_6 \vee x_7), (x_8)\}$ and $F_s = \cup_{i=1}^{8}\{(\neg x_i)\}$. Invoke WPM1 [50] on F and assume that it first processes the core $\{(\neg x_3), (\neg x_4), (\neg x_5)\}$. Afterwards the working formula F^2 is $F^2_h = F_h \cup \{\mathrm{CNF}(r_1 + r_2 + r_3 = 1)\}$ and $F^2_s = \{(\neg x_1), (\neg x_2), (\neg x_3 \vee r_1), (\neg x_4 \vee r_2), (\neg x_5 \vee r_3), (\neg x_6), (\neg x_7)(\neg x_8)\}$. Now $\mathbf{card}^2 = \{\mathrm{CNF}(r_1 + r_2 + r_3 =$

1)} and the set $\mathbf{soln}(\mathbf{card}^2)$ contains three assignments α^i, $i = 1,\ldots,3$, assigning r_i to 1 and the others to 0. By Theorem 2, the set $\kappa^2 = \{(\neg x_1),(\neg x_2),(\neg x_3 \vee r_1),(\neg x_5 \vee r_3),(\neg x_6),(\neg x_7)\}$ is a core of F^2 as each $\kappa^2|_{\alpha^i}$ is a core of F. For example, $\kappa^2|_{\alpha^1} = \{(\neg x_1),(\neg x_2),(\neg x_5),(\neg x_6),(\neg x_7)\}$. In order to use Theorem 3 to show that κ^2 is also a MUS of F^2, note that $\mathrm{R}(\kappa^2) = \{(\neg x_1),(\neg x_2),(\neg x_3),(\neg x_5),(\neg x_6),(\neg x_7)\} = \{(\neg x_1),(\neg x_2),(\neg x_3)\} \cup \{(\neg x_5),(\neg x_6),(\neg x_7)\}$, where $\{(\neg x_1),(\neg x_2),(\neg x_3)\}$ and $\{(\neg x_5),(\neg x_6),(\neg x_7)\}$ are MUSes of F. Condition 2 of Theorem 3 follows since the only MUS in $\kappa^2|_{\alpha^3}$ is $\{(\neg x_1),(\neg x_2),(\neg x_3)\}$ and the only MUS in $\kappa^2|_{\alpha^1}$ is $\{(\neg x_5),(\neg x_6),(\neg x_7)\}$.

Next we prove Theorem 3. We begin by some lemmas. Assume for each of them that CG is invoked on an instance F and that F^i is the working formula on iteration i.

Lemma 1. *Let M^i be a MUS of F^i and $C^i \in M^i$. There is an $\alpha \in \mathbf{soln}(\mathbf{card^i})$ s.t. $\mathrm{R}(C^i)$ is necessary for $M^i|_\alpha$.*

Proof. By Theorem 2, $M^i|_{\alpha'}$ is a core of F for all $\alpha' \in \mathbf{soln}(\mathbf{card}^i)$. Hence it suffices to show that $M^i|_\alpha \setminus \mathrm{R}(C^i)$ is not a core for some α. Consider the assignment $(\tau{:}\alpha)$ satisfying $F_h^i \wedge (M^i \setminus \{C^i\})$, guaranteed to exist as M^i is a MUS of F^i. Now τ satisfies $F_h \wedge (M^i \setminus \{C^i\})|_\alpha = F_h \wedge (M^i|_\alpha \setminus \mathrm{R}(C^i))$ as required. □

Corollary 3. *For any MUS M^i of F^i, $\mathrm{R}(M^i) \subseteq \bigcup \mathsf{mus}(F)$.*

Corollary 4. *For any MUS M^i of F^i, there is an irreducible $\Upsilon \subseteq \mathsf{mus}(F)$ s.t. $\mathrm{R}(M^i) = \bigcup_{M \in \Upsilon} M$.*

Proof. Take Υ as the smallest collection of MUSes of F for which $\mathrm{R}(M^i) \subseteq \bigcup_{M \in \Upsilon} M$; by Corollary 3 such a collection exists. We claim that $\bigcup_{M \in \Upsilon} M \subseteq \mathrm{R}(M^i)$, from which irreducibility follows directly by minimality of Υ. Fix an arbitrary $C_e \in M$ in some $M \in \Upsilon$. By minimality of Υ, there is a clause $C^i \in M^i$ for which the only MUS of Υ containing $\mathrm{R}(C^i)$ is M. By Lemma 1, there exists a β for which $\mathrm{R}(C^i)$ is necessary for $M^i|_\beta$. As $M^i|_\beta \subseteq \mathrm{R}(M^i) \subseteq \bigcup_{M \in \Upsilon} M$ and the only MUS in Υ containing $\mathrm{R}(C^i)$ is M, we have $C_e \in M \subseteq M^i|_\beta \subseteq \mathrm{R}(M^i)$, establishing $C_e \in \mathrm{R}(M^i)$ and $\bigcup_{M \in \Upsilon} M \subseteq \mathrm{R}(M^i)$. □

We are now ready to prove Theorem 3.

Proof (of Theorem 3). A collection $\Upsilon \subseteq \mathsf{mus}(F)$ satisfying condition 1 exists by Corollary 4. For condition 2, we use the fact that the set Υ is irreducible. Let $M \in \Upsilon$ be arbitrary. Similarly to the proof of Corollary 4, we can find a $C^i \in M^i \in \Upsilon$ and $\alpha \in \mathbf{soln}(\mathbf{card}^i)$ s.t $\mathrm{R}(C^i) \notin M'$ for any other $M' \in \Upsilon$ and $\mathrm{R}(C^i)$ is necessary for $M^i|_\alpha$, implying that the only MUS in $M^i|_\alpha$ is M. Finally, condition 3 follows from M^i being a core of F^i and Theorem 2.

What remains is to show that subset $M^i \subseteq F_s^i$ satisfying conditions 1–3 is a MUS of F^i. By condition 3 and Theorem 2, M^i is a core of F^i. Hence we only need to show that it is minimally unsatisfiable, i.e., $F_h^i \wedge (M^i \setminus \{C^i\})$ is satisfiable for all $C^i \in M^i$. Fix $C^i \in M^i$ and let Υ be the collection of MUSes of

F for which $R(M^i) = \bigcup_{M \in \Upsilon} M$. Consider any MUS $M_C \in \Upsilon$ s.t. $R(C^i) \in M_C$. By condition 2, there is an $\alpha \in \mathbf{soln}(\mathbf{card}^i)$ for which the only MUS (of F) in $M^i|_\alpha \subseteq R(M^i)$ is M_C. For such α, $F_h \wedge M^i|_\alpha \setminus \{R(C_i)\}$ is satisfied by some τ. Hence $(\tau{:}\alpha)$ satisfies $F_h \wedge \mathbf{card}^i \wedge (M^i \setminus \{C^i\}) = F_h^i \wedge (M^i \setminus \{C^i\})$. $\qquad\square$

Finally, we note that each condition in Theorem 3 is necessary.

Example 4. Consider again the MaxSAT instance F from Example 3. The set $\{(\neg x_1), (\neg x_2), (\neg x_3 \vee r_1)\}$ is an example of a non-MUS of F^1 satisfying conditions 1–2 and the set $\{(\neg x_1), (\neg x_2), (\neg x_3 \vee r_1), (\neg x_5 \vee r_3), (\neg x_6), (\neg x_7), (\neg x_8)\}$ is an example of a non-MUS of F^1 satisfying conditions 1 and 3.

6.2 Results on Core Trace Lengths

We proceed with proofs on the number of iterations for CG. With respect to best-case, preprocessing does not affect the lengths of CG^{mus} traces significantly.

Proposition 5.
For any MaxSAT instance F, $\mathsf{minlen}(CG^{\mathsf{mus}}, F) = \mathsf{minlen}(CG^{\mathsf{mus}}_{pre}, F)$.

Proof. We show that a $T_m = (m^1, \ldots, m^n)$ is a CG^{mus} trace on F iff it is a CG^{mus}_{pre} trace on F. We prove the left-to-right direction, the other is similar. We will show that there is an execution of CG^{mus}_{pre} on F for which the ith MUS extracted is m^i and which terminates only after extracting all MUSes of T_m. The termination follows from no proper subset of a CG^{mus} trace being a core trace in itself.

We show that each m^i is a MUS of $\mathsf{pre}(F)^i$ by induction. By Corollary 1, m^1 is a MUS of $\mathsf{pre}(F)$. Assume that CG^{mus} has extracted and processed the MUSes $(m^1, \ldots, m^{i-1})$ from $\mathsf{pre}(F)$ and consider the ith iteration. As m^i is a MUS of F^i, by Theorem 3 there is an $\Upsilon \subseteq \mathsf{mus}(F)$ s.t. $R(m^i) = \cup_{m \in \Upsilon} m$. For $m^i \in \mathsf{mus}(\mathsf{pre}(F)^i)$, we show that Υ satisfies the conditions of Theorem 3 in $\mathsf{pre}(F)$ as well. By Corollary 1, each $m \in \Upsilon$ is a MUS of $\mathsf{pre}(F)$. For the other two conditions, note that by induction, the set of cardinality constraints $\mathbf{card}^i_p$ added to $\mathsf{pre}(F)$ after processing the MUSes $m^1, \ldots, m^{i-1}$ is the same as the set $\mathbf{card}^i$ added to F after processing the same sequence of MUSes. Hence $\alpha \in \mathbf{soln}(\mathbf{card}^i_p)$ iff $\alpha \in \mathbf{soln}(\mathbf{card}^i)$, which implies the two other conditions of Theorem 3. $\qquad\square$

Next we show that some shortest execution of CG is also an execution of CG^{mus}.

Proposition 6.
For any MaxSAT instance F, $\mathsf{minlen}(CG^{\mathsf{mus}}, F) \leq \mathsf{minlen}(CG, F)$ and $\mathsf{minlen}(CG^{\mathsf{mus}}_{pre}, F) \leq \mathsf{minlen}(CG_{pre}, F)$.

Proof. (Sketch) We prove $\mathsf{minlen}(CG^{\mathsf{mus}}, F) \leq \mathsf{minlen}(CG, F)$; the same proof works for $\mathsf{minlen}(CG^{\mathsf{mus}}_{pre}, F) \leq \mathsf{minlen}(CG_{pre}, F)$ as well. Let $T = (\kappa^1, \ldots, \kappa^n)$ be a CG trace on F. We construct a CG^{mus} trace $T_m = (m^1, \ldots, m^k)$ on F of

at most the same length recursively. For intuition, on each iteration i CG$^{\mathsf{mus}}$ processes a subset of the clauses CG would have processed on the ith iteration of the execution corresponding to T. Hence, if $\mathbf{card}_m^i$ and $\mathbf{card}^i$ are the set of cardinality constraints added to F by the ith iteration on the execution corresponding to T_m and T, respectively, then any $\alpha \in \mathbf{soln}(\mathbf{card}_m^i)$ can be extended to a solution to $\mathbf{card}^i$ by assigning the remaining variables to 0.

Let m^1 be an MUS of F contained in κ^1. Assume that CG$^{\mathsf{mus}}$ has extracted the MUSes m^j for $j = 1, \ldots, i-1$ s.t each $m^j \subseteq \kappa^j$. Consider the ith iteration and the current working formula F_m^i. As κ^i is a core of F^i, the ith working formula on the execution corresponding to T, by Theorem 2 $\kappa^i|_\beta$ is a core of F for all $\beta \in \mathbf{soln}(\mathbf{card}^i)$. Hence $\kappa^i|_\beta$ is also a core of F for all $\beta \in \mathbf{card}_m^i$. Applying Theorem 2 gives that κ^i is a core of F_m^i. Hence it also contains a MUS m^i of F_m^i. For termination of CG$^{\mathsf{mus}}$, note that $\min_{C^i \in \kappa^i}\{w(C^i)\} \leq \min_{C^i \in m^i}\{w(C^i)\}$ for every i. Since $\sum_{i=1}^n \min_{C^i \in \kappa^i}\{w(C^i)\} = \text{COST}(F)$, termination of CG$^{\mathsf{mus}}$ occurs at the latest after n iterations on the execution corresponding T_m.   □

Finally, we show that the shortest executions of CG and CG$_{\mathsf{pre}}$ are of the same length.

Corollary 5. *For any MaxSAT instance F, $\mathsf{minlen}(CG, F) = \mathsf{minlen}(CG_{\mathsf{pre}}, F)$.*

Proof. Proposition 6 and Observation 1 imply $\mathsf{minlen}(CG, F) = \mathsf{minlen}(CG^{\mathsf{mus}}, F)$ and $\mathsf{minlen}(CG_{\mathsf{pre}}^{\mathsf{mus}}, F) = \mathsf{minlen}(CG_{\mathsf{pre}}, F)$. Together with Proposition 5 we obtain $\mathsf{minlen}(CG, F) = \mathsf{minlen}(CG^{\mathsf{mus}}, F) = \mathsf{minlen}(CG_{\mathsf{pre}}^{\mathsf{mus}}, F) = \mathsf{minlen}(CG_{\mathsf{pre}}, F)$.   □

We move on to worst-case results for CG. We begin by showing that valid executions of CG$_{\mathsf{pre}}$ are also valid executions of CG.

Proposition 7.
For any MaxSAT instance F, $\mathsf{maxlen}(CG, F) \geq \mathsf{maxlen}(CG_{\mathsf{pre}}, F)$.

Proof. We show that a CG$_{\mathsf{pre}}$ trace $T = (\kappa^1, \ldots, \kappa^n)$ on F is also a CG trace on F. The termination of CG only after n iterations follows from the cost-preserving properties of preprocessing and Observation 4. We show that each κ^i is a valid core of F^i by induction. The case $i = 1$ follows from $\mathsf{pre}(F)_s \subseteq F_s$ and Corollary 1. Assume next that all κ^j for $j < i$ have been cores of F^j and consider κ^i. By Theorem 2, $\kappa^i|_\beta$ is a core of $\mathsf{pre}(F)$ for all $\beta \in \mathbf{soln}(\mathbf{card}_p^i)$, where $\mathbf{card}_p^i$ is the set of cardinality constraints added to $\mathsf{pre}(F)$ after processing cores $\kappa^1, \ldots, \kappa^{i-1}$. By induction, this set is exactly the same as set of cardinality constraints $\mathbf{card}^i$ added to F after processing the same cores. As any core of $\mathsf{pre}(F)$ is a core of F, it follows that $\kappa^i|_\beta$ is a core of F for all $\beta \in \mathbf{soln}(\mathbf{card}^i)$. We conclude that κ^i is a core of F^i.   □

Finally, two families of instances witness the $\nrightarrow$ edges in Fig. 4 for CG.

Proposition 8.
There is a family of MaxSAT instances $F(n)$ with $\mathcal{O}(n)$ soft clauses s.t. $\mathsf{maxlen}(CG, F(n)) \geq n$ and $\mathsf{maxlen}(CG^{\mathsf{mus}}, F(n)) = \mathsf{maxlen}(CG_{\mathsf{pre}}, F(n)) = 1$.

Proof. (Sketch) Consider again the instance $F(n)$ constructed in the proof of Proposition 3. We showed that $\mathsf{maxlen}(\mathsf{HS}^{\mathsf{mus}}, F) = \mathsf{maxlen}(\mathsf{HS}_{\mathsf{pre}}, F) = 1$. This also holds for CG. A linear-length CG core trace $(\kappa^1, \ldots, \kappa^n)$, on F can be constructed iteratively as follows: $\kappa^1 = \{(\neg x), (\neg y), (\neg z_1)\}$ and $\kappa^i = \{(\neg x)^c_{i-1}, (\neg y)^c_{i-1}, (\neg z_i)\}$ where $(\neg x)^c_{i-1}$ and $(\neg y)^c_{i-1}$ are duplicates of the original clauses added on iteration $i-1$. The existence of such duplicates for all n iterations follows from $w((\neg x)) = w((\neg y)) = n$ and $w((\neg z_i)) = 1$. The termination of CG after the nth iteration follows from Observation 4 as the smallest weight among the clauses in each κ^i is 1.   □

Proposition 9. *There is a family of MaxSAT instances $F(n)$ with $\mathcal{O}(n)$ soft clauses s.t.* $\mathsf{maxlen}(CG_{\mathsf{pre}}, F(n)) \geq n$ *and* $\mathsf{maxlen}(CG^{\mathsf{mus}}_{\mathsf{pre}}, F) = 1$.

Proof. (Sketch) $F(n)$ is the same as for HS and the proof follows Proposition 4. A linear-length CG core trace can be constructed similarly to Proposition 8 by replacing clauses in the linear-length HS trace from Proposition 4 with duplicates of original clauses where required.   □

7   Conclusions

We formally analyzed the effect of SAT-based preprocessing, as well as core minimization, on the performance of core-guided MaxSAT solvers. As a main result, we showed that SAT-based preprocessing has no effect on the best-case number of iterations required by the solvers but can improve on the worst-case. In terms of best-case performance, the potential benefits of applying SAT-based preprocessing in conjunction with core-guided MaxSAT solvers are thus in principle—assuming optimal search heuristics—solely in speeding up individual SAT solver calls made during MaxSAT search. Simultaneously, our analysis also revealed an analogous result on the impact of core minimization in core-guided MaxSAT solvers. Our results motivate further work on developing MaxSAT-specific preprocessing techniques capable of affecting the MaxSAT algorithms on a more general level. In contrast, SAT-based preprocessing does in cases have a positive effect on the worst-case number of iterations. Of independent interest, we established a formal characterization of how the underlying MUS structure is altered by iterative revisions performed by CG solvers on MaxSAT instances (Theorem 3), thus sharpening the main results of [53].

Appendix

A   Proof of Proposition 1

(1) If an optimal solution τ to F assigns $\tau(C) = 0$, then an optimal solution τ^P to F_P has to assign $F_P(l_C) = 1$. Similarly, if $\tau(C) = 1$, then τ^P can assign $\tau^P(l_C) = 0$.

(2) We sketch the conversion of an $\mathcal{A}$ core trace $T_P = (\kappa_P^1, \ldots, \kappa_P^n)$ on F_P into a core trace $T = (\kappa^1, \ldots, \kappa^n)$ on F, the other direction is similar. For $\mathcal{A} = $ HS, every κ_P^i is a core of F_P. The corresponding core trace of F is obtained by exchanging each $\kappa_P^i = \{(\neg l_{C_i}) \mid i = 1, \ldots, n\}$ with $\kappa^i = \{C_i \mid i = 1, \ldots, n\}$. Now κ_P^i is a core of F_P iff κ^i is a core of F. To see this, note that if κ^i is not a core of F, then it can be satisfied by some assignment τ. The same τ extended by setting all l_{C_i} variables to 0 to satisfies both κ_P^i and the hard clauses $\{C_1 \vee l_{C_1}, \ldots, C_n \vee l_{C_n}\}$. Hence κ_P^i is not a core of F_P either. A similar argument shows the other direction. Finally the termination of HS after n iterations follows by a similar argument showing that $F \setminus hs$ is satisfiable for some $hs = \{C_1, \ldots, C_i\}$ iff $F^P \setminus hs^P$ is satisfiable for $hs^P = \{(\neg l_{C_1}), \ldots, (\neg l_{C_i})\}$. Hence the trace $T = (\kappa^1, \ldots, \kappa^n)$ is a HS trace on F of the same length as T_P.

For $\mathcal{A} = $ CG the argument is similar but inductive. To form a CG trace T on F, every occurrence of a $(\neg l_{C_i})$ in a clause $C^i \in \kappa_P^i$ is replaced by C_i to form a core κ^i of F^i. For $i > 0$, each such C^i may have been augmented with blocking variables, i.e., $C^i = (\neg l_{C_i} \vee \bigvee b)$ for some set of blocking variables. However, the substitution $(\neg l_{C_i} \vee \bigvee b) \rightarrow C_i \vee \bigvee b$ is still valid as, by induction, if CG adds $\bigvee b$ to $(\neg l_{C_i})$ on the execution corresponding to T_P, then it also adds $\bigvee b$ to C_i on the execution corresponding to T. $\qquad\square$

References

1. Park, J.D.: Using weighted MAX-SAT engines to solve MPE. In: Proceedings of the AAAI, pp. 682–687. AAAI Press/The MIT Press (2002)
2. Chen, Y., Safarpour, S., Veneris, A.G., Marques-Silva, J.P.: Spatial and temporal design debug using partial MaxSAT. In: Proceedings of the 19th ACM Great Lakes Symposium on VLSI, pp. 345–350. ACM (2009)
3. Chen, Y., Safarpour, S., Marques-Silva, J., Veneris, A.G.: Automated design debugging with maximum satisfiability. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(11), 1804–1817 (2010)
4. Argelich, J., Berre, D.L., Lynce, I., Marques-Silva, J.P., Rapicault, P.: Solving linux upgradeability problems using boolean optimization. In: Proceedings of the LoCoCo, EPTCS, vol. 29, pp. 11–22 (2010)
5. Lynce, I., Marques-Silva, J.: Restoring CSP satisfiability with MaxSAT. Fundam. Inform. **107**(2–3), 249–266 (2011)
6. Zhu, C., Weissenbacher, G., Malik, S.: Post-silicon fault localisation using maximum satisfiability and backbones. In: Proceedings of the FMCAD, pp. 63–66. FMCAD Inc. (2011)
7. Jose, M., Majumdar, R.: Cause clue clauses: error localization using maximum satisfiability. In: Proceedings of the PLDI, pp. 437–446. ACM (2011)
8. Morgado, A., Liffiton, M., Marques-Silva, J.: MaxSAT-based MCS enumeration. In: Biere, A., Nahir, A., Vos, T. (eds.) HVC. LNCS, vol. 7857, pp. 86–101. Springer, Heidelberg (2013)
9. Guerra, J., Lynce, I.: Reasoning over biological networks using maximum satisfiability. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 941–956. Springer, Heidelberg (2012)
10. Zhang, L., Bacchus, F.: MAXSAT heuristics for cost optimal planning. In: Proceedings of the AAAI. AAAI Press (2012)

11. Ansótegui, C., Izquierdo, I., Manyà, F., Torres-Jiménez, J.: A Max-SAT-based approach to constructing optimal covering arrays. In: Proceedings of the CCIA, Frontiers in Artificial Intelligence and Applications, vol. 256, pp. 51–59. IOS Press (2013)
12. Ignatiev, A., Janota, M., Marques-Silva, J.: Towards efficient optimization in package management systems. In: Proceedings of the ICSE, pp. 745–755. ACM (2014)
13. Berg, J., Järvisalo, M., Malone, B.: Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In: Proceedings of the AISTATS, JMLR Workshop and Conference Proceedings, vol. 33, pp. 86–95 (2014). www.JMLR.org
14. Fang, Z., Li, C., Qiao, K., Feng, X., Xu, K.: Solving maximum weight clique using maximum satisfiability reasoning. In: Proceedings of the ECAI, Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 303–308. IOS Press (2014)
15. Berg, J., Järvisalo, M.: SAT-based approaches to treewidth computation: an evaluation. In: Proceedings of the ICTAI, pp. 328–335. IEEE Computer Society (2014)
16. Marques-Silva, J., Janota, M., Ignatiev, A., Morgado, A.: Efficient model based diagnosis with maximum satisfiability. In: Proceedings of the IJCAI, pp. 1966–1972. AAAI Press (2015)
17. Berg, J., Järvisalo, M.: Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. Artificial Intelligence (2015, in press)
18. Wallner, J.P., Niskanen, A., Järvisalo, M.: Complexity results and algorithms for extension enforcement in abstract argumentation. In: Proceedings of the AAAI. AAAI Press (2016)
19. Li, C., Manyà, F.: MaxSAT, hard and soft constraints. In: Handbook of Satisfiability, pp. 613–631. IOS Press (2009)
20. Ansótegui, C., Bonet, M., Levy, J.: SAT-based MaxSAT algorithms. Artif. Intell. **196**, 77–105 (2013)
21. Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: a survey and assessment. Constraints **18**(4), 478–534 (2013)
22. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the STOC, pp. 151–158. ACM (1971)
23. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability: Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)
24. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: a partial Max-SAT solver. J. Satisfiability Boolean Model. Comput. **8**(1/2), 95–100 (2012)
25. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MAXSAT. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 166–181. Springer, Heidelberg (2013)
26. Belov, A., Morgado, A., Marques-Silva, J.: SAT-based preprocessing for maxsat. In: Middeldorp, A., Voronkov, A., McMillan, K. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 96–111. Springer, Heidelberg (2013)
27. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: a modular maxsat solver. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 438–445. Springer, Heidelberg (2014)
28. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: Proceedings of the AAAI, pp. 2717–2723. AAAI Press (2014)
29. Bjørner, N., Narodytska, N.: Maximum satisfiability using cores and correction sets. In: Proceedings of the IJCAI, pp. 246–252. AAAI Press (2015)

30. Berg, J., Saikko, P., Järvisalo, M.: Improving the effectiveness of SAT-based pre-processing for MaxSAT. In: Proceedings of the IJCAI, pp. 239–245. AAAI Press (2015)
31. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 564–573. Springer, Heidelberg (2014)
32. Ansótegui, C., Gabàs, J.: Solving (weighted) partial MaxSAT with ILP. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013. LNCS, vol. 7874, pp. 403–409. Springer, Heidelberg (2013)
33. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
34. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Miller, D., Sattler, U., Gramlich, B. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 355–370. Springer, Heidelberg (2012)
35. Lagniez, J.M., Marquis, P.: Preprocessing for propositional model counting. In: Proceedings of the AAAI, pp. 2688–2694. AAAI Press (2014)
36. Li, C.M., Manyà, F., Mohamedou, N., Planes, J.: Exploiting cycle structures in Max-SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 467–480. Springer, Heidelberg (2009)
37. Argelich, J., Li, C.-M., Manyà, F.: A preprocessor for Max-SAT solvers. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 15–20. Springer, Heidelberg (2008)
38. Bonet, M.L., Levy, J., Manyà, F.: Resolution for Max-SAT. Artif. Intell. **171**(8–9), 606–618 (2007)
39. Heras, F., Marques-Silva, J.: Read-once resolution for unsatisfiability-based Max-SAT algorithms. In: Proceedings of the IJCAI, pp. 572–577. AAAI Press (2011)
40. Berg, J., Saikko, P., Järvisalo, M.: Re-using auxiliary variables for maxsat pre-processing. In: Proceedings of the ICTAI, pp. 813–820. IEEE (2015)
41. Krentel, M.W.: The complexity of optimization problems. J. Comput. Syst. Sci. **36**(3), 490–509 (1988)
42. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proceedings of the AAAI. AAAI Press (2011)
43. Ignatiev, A., Morgado, A., Manquinho, V.M., Lynce, I., Marques-Silva, J.: Progression in maximum satisfiability. In: ECAI 2014, pp. 453–458. IOS Press (2014)
44. Kullmann, O., Marques-Silva, J.: Computing maximal autarkies with few and simple oracle queries (2015). CoRR abs/1505.02371
45. Janota, M., Marques-Silva, J.: On the query complexity of selecting minimal sets for monotone predicates. Artif. Intell. **233**, 73–83 (2016)
46. Ansótegui, C., Gabàs, J., Levy, J.: Exploiting subproblem optimization in SAT-based MaxSAT algorithms. J. Heuristics **22**(1), 1–53 (2016)
47. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
48. Belov, A., Järvisalo, M., Marques-Silva, J.: Formula preprocessing in MUS extraction. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 108–123. Springer, Heidelberg (2013)
49. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006)

50. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 495–508. Springer, Heidelberg (2009)
51. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 427–440. Springer, Heidelberg (2009)
52. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 531–548. Springer, Heidelberg (2014)
53. Bacchus, F., Narodytska, N.: Cores in core based MaxSat algorithms: an analysis. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 7–15. Springer, Heidelberg (2014)
54. Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 247–262. Springer, Heidelberg (2013)
55. Saikko, P., Berg, J., Järvisalo, M.: LMHS: a SAT-IP hybrid MaxSAT solver. In: Creignou, N., Le Berre, D., Le Berre, D., Le Berre, D., Le Berre, D., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 539–546. Springer, Heidelberg (2016). doi:10.1007/978-3-319-40970-2_34

Multiobjective Optimization
by Decision Diagrams

David Bergman[1(✉)] and Andre A. Cire[2]

[1] Department of Operations and Information Management,
University of Connecticut, Mansfield, USA
`david.bergman@business.uconn.edu`
[2] Department of Management, University of Toronto Scarborough, Toronto, Canada
`acire@utsc.utoronto.ca`

Abstract. In this paper we present a technique for solving multiobjective discrete optimization problems using decision diagrams. The proposed methodology is related to an algorithm designed for multiobjective optimization for dynamic programming, except utilizing decision diagram theory to reduce the state space, which can lead to orders of magnitude performance gains over existing algorithms. The decision diagram-based technique is applied to knapsack, set covering, and set partitioning problems, exhibiting improvements over state-of-the-art general-purpose multiobjective optimization algorithms.

Keywords: Decision diagrams · Multiobjective optimization · Multicriteria decision making · Multicriteria shortest path

1 Introduction

Automated decision making, by its very nature, requires the consideration of a multitude of objectives. Multiobjective optimization, also known as multiobjective programming, vector optimization, multi-criteria optimization, multi-attribute optimization or Pareto optimization, has a rich history, dating back to the emergence of rigorous mathematical programming [21]. Several books have been written on the topic [11,14,28], and many in-depth surveys [12,13,15,34,37].

The present paper concerns *multiobjective discrete optimization problems* (MODO), where the variables of the problem are discrete and the number of objectives is $p \geq 2$. An *efficient solution* to a MODO is one in which there is no other solution that improves, simultaneously, on each of the objectives. The vector corresponding to the objectives of an efficient solution is called a *nondominated solution*. The set of efficient solutions is known as the *efficient set* and the set of all nondominated solution is known as the *nondominated* set. The goal of a MODO, for the purposes of this paper, is to enumerate the nondominated set.

Perhaps the most commonly studied technique for identifying all nondominated solutions for MODOs is the ϵ-*constraint* method [18], having well-known properties for more than two objectives [11]. The ϵ-constraint method was first

used by Laumanns et al. [24], where an adaptive search over weighted objectives was proposed. Özlen et al. [27,29] and Kirlik and Sayın [23] provide algorithmic improvements and are the basis for the comparison in the computational results presented in this paper. Other utilized scalarization techniques include Benson's method [3] (a combination of the weight-sum technique and the ϵ-constraint method) and the augmented weighted Chebychev method [32]. The main drawback of these techniques is that a discrete optimization problem, often NP-hard in practice, must be solved several times so as to enumerate the necessary scalarizations, leading to the bottleneck of their procedures.

In this paper, we propose an alternative method that creates a single *binary decision diagram* (BDD) [1,9,10,25] which represents all feasible solutions to the problem as a directed acyclic graph, and then employs *multicriteria shortest path problem* (MSP) algorithms to enumerate the nondominated set. This technique therefore transforms the problem into that of (1) finding the exact BDD for the constraint set of the MODO, and (2) using MSP algorithms to find the set of nondominated solutions.

The utilization of decision diagrams in optimization is recent and focusses on the use of BDDs for a variety of purposes [5]. In this research stream, top-down compilation methods for set covering problems [6,7] and set packing problems [4,6] have been investigated, along with methods for creating BDDs for knapsack problems [2]. On the other hand, published articles on MSP are abundant. Research on MSP started in the 1970s [19,35] and has typically concentrated on two objectives (see Raith and Ehrgott [30] for details). A survey on exact methods for the MSP is provided by Garroppo et al. [17].

Our methodology is primarily based on the work by Loui [26], who proposes a multidimensional labeling technique for dynamic programming (DP) models with stochastic or multidimensional weights. Such techniques have been considered in several papers [30] and are known in the DP community (see, e.g., Bertsekas [8], Sect. 2.3.4), often being applied to multiobjective knapsack algorithms, as by Figueira et al. [31]. Nonetheless, labeling algorithms are typically prohibitive in practice due to the curse of dimensionality of DP models, as the state space often grows too quickly in practice.

BDDs are closely related to dynamic programming and, as in DPs, they also reduce discrete optimization to shortest path problems. However, it has been shown that the state-space associated with a BDD can be much more compact than that of a similar DP model [20], which is key to our methodology. Our major contribution, hence, is the introduction of a MSP labeling algorithms for decision diagrams. This yields an alternative, simple method for enumerating the nondominated set of a multiobjective problem. The technique is applicable to a number of discrete optimization problems, and our numerical study indicates that it can be orders of magnitude faster than state-of-the-art techniques.

2 Multiobjective Discrete Optimization

A MODO $\mathcal{M}$ is specified by a set of p objective functions $f^j : \mathbb{R}^n \rightarrow \mathbb{R}$, for $j = 1, \ldots, p$, and a *feasible set* $\mathcal{X}$. In this paper it is assumed that

each objective function is additively separable or, more simply, linear, so that $f^j(x) = \sum_{i=1}^{n} c_i^j x_i$. The feasible set $\mathcal{X}$ is discrete — this paper will focus on *binary optimization problems* so that $\mathcal{X} \subseteq \mathbb{B}^n$, although the techniques are easily generalizable. Each $x \in \mathcal{X}$ is said to be a *feasible solution*.

Let f be the set of objective functions $f : \mathbb{R}^n \to \mathbb{R}^p$ (assumed to be maximized unless otherwise specified). Each solution $x \in \mathcal{X}$ is mapped through the function f into a corresponding objective vector $y = f(x) \in \mathbb{R}^p$. The set of objective vectors $\mathcal{Y} = \{f(x) : x \in \mathcal{X}\}$ resulting from feasible solutions is the *objective space*. A solution x^* is called an *efficient solution* if there exists no other feasible solution x' such that, for all j, $f^j(x') \geq f^j(x^*)$ with $f^{j'}(x') > f^{j'}(x^*)$ for some j'. For an efficient solution x^* the vector $f(x^*)$ is referred to as a *nondominated solution*. The *efficient set*, denoted by $\mathcal{X}_\mathrm{E}$, is the set of all efficient solutions and its image, denoted by $\mathcal{Y}_\mathrm{N} = \{y : y = f(x) \text{ for some } x \in \mathcal{X}_\mathrm{E}\}$, is the *nondominated set*. The (typical) goal of MODO and the focus of this paper is to obtain $\mathcal{Y}_\mathrm{N}$.

3   Binary Decision Diagrams

A *binary decision diagrams* BDD $B = (n, U, A, \ell, d)$ is a *layered-acyclic digraph* composed of node set U and arcs A. The mapping $\ell : U \to \{1, 2, \ldots, n+1\}$ partitions the nodes into $n+1$ layers $L_i := \{u \in U : \ell(u) = i\}, i = 1, 2, \ldots, n+1$. Layers L_1 and L_{n+1} have cardinality one, with the single nodes in these layers denoted by *root* $\mathbf{r}$ and *terminal* $\mathbf{t}$, respectively. Each arc $a \in A$ leaves a *tail* node $t(a)$ and enters a *head* node $h(a)$, where $t(a), h(a) \in U$. It is assumed that $\ell(h(a)) = \ell(t(a)) + 1$, so that each arc connects nodes in adjacent layers. Also, each node u has at most one out-directed arc a with *arc-domain* $d \in \{0, 1\}$. An arc leaving node u is denoted by $a_0(u)$ if $d = 0$ and by $a_1(u)$ otherwise. The *width* of a layer is $w(L_i) := |L_i|$, and the *width* of B is $w(B) := \max_{i \in \{1, \ldots, n+1\}} w(L_i)$. Finally, the *size* of a BDD is $|U|$.

A BDD represents a set of binary vectors in the following way. Each arc-specified path $p = (a_1, a_2, \ldots, a_k)$ represents the vector $x(p) = (d(a_1), d(a_2), \ldots, d(a_k))$. Any path from $\mathbf{r}$ to $\mathbf{t}$ thereby corresponds to a vector in $\mathbb{B}^n$. Let $\mathcal{P}(B)$ be the set of arc-specified paths from $\mathbf{r}$ to $\mathbf{t}$. Define $\mathbf{Sol}(B)$ as the set of binary vectors (called *solutions*) corresponding to arc-specified $\mathbf{r} - \mathbf{t}$ paths:

$$\mathbf{Sol}(B) = \{x(p) \in \mathbb{B}^n : p \in \mathcal{P}(B)\}.$$

BDDs can be used to represent the feasible set of a MODO $\mathcal{M}$ through relating $\mathbf{Sol}(B)$ with the set of feasible solutions $\mathcal{X}$ of $\mathcal{M}$. BDD B is said to be an *exact BDD* for $\mathcal{M}$ if $\mathbf{Sol}(B) = \mathcal{X}$.

Fix $u, v \in U$ for which $\ell(u) < \ell(v)$. Let $B_{u,v}$ be the BDD obtained by removing from B any nodes and arcs that do not lie on any directed path from u to v. A BDD is said to be *reduced* if for all $i = 1, \ldots, n$ and any two nodes $u, u' \in L_i$, the BDDs $B_{u,\mathbf{t}}$ and $B_{u',\mathbf{t}}$ are such that $\mathbf{Sol}(B_{u,\mathbf{t}}) \neq \mathbf{Sol}(B_{u',\mathbf{t}})$.

It is well-known that for any set of solutions $\mathcal{X}$, with a specified ordering of the variables given, there is a unique reduced BDD [9]. The reduction of a BDD can be performed efficiently by a simple bottom-up $O(|U| \log(|U|))$ algorithm [36].

It often has dramatic effects on the size of the BDD and the calculation of the nondominated set.

Suppose $\tilde{\mathcal{X}} \subseteq \mathbb{B}^4$ is given by the following set of solutions

$$\tilde{\mathcal{X}} = \{(0,0,0,0)(0,0,0,1)(0,0,1,0)(0,1,0,0)(0,1,0,1)(0,1,1,0)$$
$$(1,0,0,0)(1,0,0,1)(1,0,1,0)(1,1,0,0)(1,1,1,0)\}$$

Consider the BDD $\tilde{B}$, depicted in Fig. 1. Dashed/solid arcs correspond to arcs with arc-domain 0/1. $\tilde{B}$ is an exact BDD for $\tilde{\mathcal{X}}$ — each path from $\mathbf{r}$ to $\mathbf{t}$ corresponds to a solution in $\tilde{\mathcal{X}}$ and vice versa so that $\mathbf{Sol}(\tilde{B}) = \tilde{\mathcal{X}}$.

We refer the reader to the work by Bergman et al. [5] for BDD compilation procedures for general optimization problems.
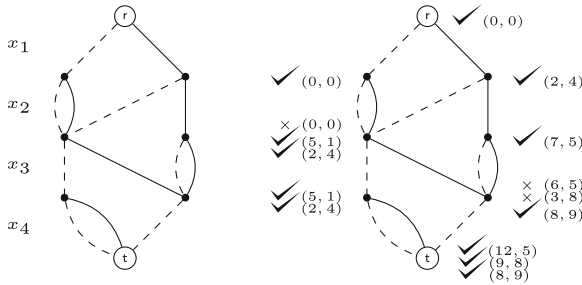
Fig. 1. Exact BDD for $\tilde{\mathcal{X}}$

4   Determining the Nondominated Set

In this section we propose a technique which generalizes previous works on single-objective optimization problems using BDDs [5]. Namely, suppose a MODO $\mathcal{M}$ has one objective function (i.e., $p = 1$) with $f^1(x) = \sum_{i=1}^{n} c_i^1 x_i$, and assume $\mathcal{X}$ is represented by a BDD B. To optimize f^1, we associate an *arc-value* $v(a)$ with each arc $a \in L_i$, $i = 1, \ldots, n$, where $v(a) = c_i^1$ if $d(a) = 1$, and $v(a) = 0$ otherwise. Any longest $\mathbf{r}$ to $\mathbf{t}$ path p with respect to v yields an optimal solution $x(p)$, with the corresponding optimal value equal to the length of the path.

Let $\mathcal{M}$ be a MODO with $p \geq 2$. Given an exact BDD B for $\mathcal{X}$, we apply Algorithm 1 to determine $\mathcal{Y}_{\mathrm{N}}$. The algorithm traverses B in a top-down fashion and associates a *state* s with each node u, representing the set of nondominated solutions in $\mathbb{R}^{\ell(u)-1}$ contained in $B_{\mathbf{r},u}$; at the end of the execution, $s(\mathbf{t}) = \mathcal{Y}_{\mathrm{N}}$.

The algorithms initializes $s(u)$ for each node $u \in U \backslash \{\mathbf{r}\}$ to $\emptyset$, and initializes $s(\mathbf{r})$ to $\{\mathbf{0}\}$ (the p-dimensional vector with 0s in each coordinate). Having set the states of each node in L_i the algorithm proceeds to determine the states for the nodes in L_{i+1}. For each node in L_i and each arc directed out of u,

---

Algorithm 1. Find $\mathcal{Y}_N$ for MODO $\mathcal{M}$

1: **procedure** FINDNDS($\mathcal{M}, B$)   ▷ $\mathcal{X}$ is the feasible set for $\mathcal{M}$ and
2:   objectives $f^j, j = 1, \ldots, p$
3:     **for all** $u \in U$ **do**
4:         $s(u) \leftarrow \emptyset$
5:     $s(\mathbf{r}) \leftarrow \{\mathbf{0}\}$ ▷ $\mathbf{0}$ is the p-dimensional 0 vector
6:     **for** $i = 1, \ldots, n$ **do**
7:         **for all** $u \in L_i$ **do**
8:             **if** $a_0(u)$ exists **then**
9:                 **for all** $s' \in s(u)$ **do**
10:                     **if** s' is not dominated in $s(h(a_0(u)))$ **then**
11:                         $s(h(a_0(u))) \leftarrow s(h(a_0(u))) \cup s'$
12:                     **for all** $s'' \in s(h(a_0(u)))$ dominated by s' **do**
13:                         $s(h(a_0(u))) \leftarrow s(h(a_0(u))) \setminus \{s''\}$
14:             **if** $a_1(u)$ exists **then**
15:                 **for all** $s' \in s(u)$ **do**
16:                     Increase state values of s' by c_i^j, for each $j = 1, \ldots, p$
17:                     **if** s' is not dominated in $s(h(a_1(u)))$ **then**
18:                         $s(h(a_1(u))) \leftarrow s(h(a_1(u))) \cup s'$
19:                     **for all** $s'' \in s(h(a_1(u)))$ dominated by s' **do**
20:                         $s(h(a_1(u))) \leftarrow s(h(a_1(u))) \setminus \{s''\}$
21:     **return** $s(\mathbf{t})$

---

the algorithm checks whether or not a nondominated solution will arise from extending solutions ending at u with the arc domain of the arc.

For each layer L_i, if the arc under consideration is a 0-arc, then any non-dominated solution will not be affected by setting $x_i = 0$. Therefore, each non-dominated solution in $s(u)$ is considered as a possible nondominated solution in $s(h(a))$. If it is nondominated, the solution is added to $s(h(a))$, and otherwise omitted. The process is analogous for the case of a 1-arc, except that any potential nondominated solution will have its objective value increased by c_i^j for each $j = 1, \ldots, p$. Finally, if a solution s' is added to the state of a node, we must verify if any solutions s'' in that same state are now dominated by s', removing them if that is the case.

Suppose, for example, that a MODO $\mathcal{M}'$ is specified with $\tilde{\mathcal{X}}$ defined as above and 2 objective functions, $f^1(x) = 2x_1 + 5x_2 + x_3 + 7x_4$, $f^2(x) = 4x_1 + x_2 + 4x_3 + 4x_4$. Figure 1 shows the result of applying Algorithm 1 to the BDD in Fig. 1. Each node u is labeled with a set of vectors in the objective space. Those marked with a ✓ compose $s(u)$ and those marked with a × are those candidates for $s(u)$ that are determined to be dominated by some solution in $s(u)$. As a concrete example, consider the node u marked with the three vectors $(0,0), (5,1)$ and $(2,4)$. $(0,0)$ is first added to $s(u)$ because of the zero-arc directed at u from the node immediately above it. Then, when the one-arc from that same node is considered, it generates the vector $(0,0) + (5,1) = (5,1)$, eliminating $(0,0)$ because $(5,1)$ dominates it. Then, the zero-arc from the right-most node on the

previous layer is examined, generating vector $(2,4) + (0,0) = (2,4)$. The set $\{(5,1),(2,4)\}$ is a collection of vectors, none of which dominate any other, and so $(2,4)$ is added to $s(u)$. At the conclusion of the algorithm, $\mathbf{t}$ is labeled with $s(\mathbf{t}) = \{(12,5),(9,8),(8,9)\}$, the set $\mathcal{Y}_N$ for $\mathcal{M}'$. The proof of correctness of Algorithm 1 follows immediately, e.g., from the results of Figueira et al. [31] and Loui [26].

5   Numerical Study

In this section we evaluate the empirical performance of the proposed multi-objective methodology on three classical optimization problems: *knapsack*, *set covering*, and *set packing*. Our key performance metric is the time to enumerate the complete set of nondominated solutions. We compare our approach with another general-purpose MODO solver developed by Kirlik and Sayın [23], an ϵ-constraint scalarization method which is currently regarded as one of the state of the art approaches for MODOs. We note in passing that we have also investigated other MODO solvers, such as the one proposed by Özlen et al. [29], but the method by Kirlik and Sayın was the best performing and numerically stable method across all solvers tested in our empirical setting.

The experiments ran on an Intel(R) Xeon(R) CPU E5-2640 v3 at 2.60 GHz with 128 GB RAM. The BDD method was implemented in C++ and compiled with GCC 4.8.4. Our source code and all tested instances are available at http://www.andrew.cmu.edu/user/vanhoeve/mdd/. The source code for the technique by Kirlik and Sayın was downloaded from http://home.ku.edu.tr/~moolibrary/ and linked with ILOG CPLEX 12.6.3 [22]. A time limit of 3,600 s was allotted in all cases.

Multicriteria Knapsack. Given n items, a capacity $W > 0$, and for each item i a weight $w_i > 0$ and p profits $v_i^1, v_i^2, \ldots, v_i^p > 0$, the multicriteria knapsack problem (MKP) is: $\max\left\{\sum_{i=1}^n v_i^j x_i, \; j = 1, \ldots, p : \sum_{i=1}^n w_i x_i \leq W, \; x \in \{0,1\}^n\right\}$.

We generated random MKP instances following the procedure by Kirlik and Sayın [23]. The values v_i^j and w_i were drawn uniformly at random from the set $\{1, \ldots, 1000\}$, and $W = \lceil 0.5 \sum_{i=1}^n w_i \rceil$. Due to the growth of the nondominated set, for $p = 3$, we considered $n = \{10, \ldots, 100\}$; for $p = 4$, $n \in \{10, \ldots, 70\}$; for $p = 5, 6$, and 7, $n \in \{20, 30, 40\}$. We generated 10 instances per pair (n, p).

Table 1 presents the average cardinalities of the nondominated sets $(|\mathcal{Y}_N|)$ and average solution times (within solved instances) for $p = 3, 4, 5$, where Kirlik and BDD denotes the method by Kirlik and Sayın [23] and the BDD technique, respectively. Figure 2(a) depicts a scatter plot comparing solution times for all instances, where the size of a point is proportional to n. The BDD method is substantially faster and more robust than Kirlik, in particular when p is large. In all cases, the time to create the BDD is only 5 % of the total BDD time, and therefore the bottleneck is the computation of Algorithm 1. Finally, Fig. 2(b) depicts a scatter plot comparing solution times of reduced and nonreduced BDDs for $n \in \{40, 50, 60\}$ and $p = 4$, emphasizing the importance of reducing the

Table 1. Average solution times (in seconds) for knapsack problems for $p = 3, 4, 5$. Number in parentheses indicate instances unsolved within the time limit (out of 10).

n	$p = 3$			$p = 4$			$p = 5$		
	$|\mathcal{Y}_N|$	Kirlik	BDD	$|\mathcal{Y}_N|$	Kirlik	BDD	$|\mathcal{Y}_N|$	Kirlik	BDD
10	9	0.10	0.01	14	0.28	0.01	22	0.04	0.01
20	37	0.85	0.03	79	7.22	0.04	241	$1,319.48^{(7)}$	0.04
30	113	4.95	0.17	397	$466.40^{(1)}$	0.26	972	(10)	0.47
40	370	24.71	0.95	1,278	$217.38^{(9)}$	2.39	4,943	(10)	18.06
50	598	50.47	3.64	3,374	(10)	27.87			
60	1,080	120.21	12.52	6,624	(10)	166.34			
70	1,325	154.50	32.69	14,696	(10)	1,164			
80	2,575	454.46	120						
90	3,847	912.47	350						
100	4,248	$1,070^{(1)}$	$551^{(1)}$						

number of nodes of the BDD before performing Algorithm 1 (which, on average, take less than a second on all tested cases). This saves approximately a half of an order of magnitude on the total computation time.

Multicriteria Set Covering and Set Packing. Let A be a 0–1 $m \times n$ constraint matrix, and let $c^1, \ldots, c^p$ be the p cost vectors in $\mathbb{R}^n$. The *multicriteria set covering problem* (MSCP) is defined as $\min \left\{ (c^j)^T x, j = 1, \ldots, p : Ax \geq 1, x \in \{0, 1\}^n \right\}$. The *multicriteria set packing problem* (MSPP) is similar to the MSCP and is

(a) Solution times.   (b) Reduced and non-reduced.
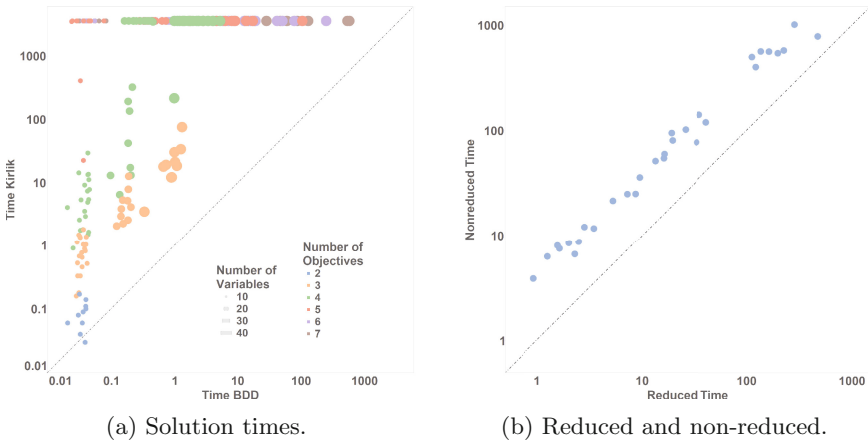
Fig. 2. (a) Solution time comparison between Kirlik and BDD (in logarithmic scale), and (b) Solution times for reduced and non-reduced BDDs for the MKP.

written as $\max\left\{(c^j)^T x, j = 1, \ldots, p : Ax \leq 1, x \in \{0,1\}^n\right\}$. The multiobjective variants pose a considerably more difficult challenge [16].

We performed experiments on random instances generated as in Stidsen et al. [33]. Specifically, we considered $n \in \{100, 150\}$, $m = n/5$, and in our case we fixed 10 variables per constraint, i.e., for every $k = 1, \ldots, m$, 10 elements of the k-th row of A were chosen to be equal to one uniformly at random. 10 instances were created per pair (n, p). The objective coefficients were generated as they were for the MKP. The reduced BDDs for the MSPP and MSCP were compiled according to Bergman et al. [5,7].

Table 2 presents the average cardinalities of the nondominated sets and the average solution times. As before, BDD is substantially faster and more robust than Kirlik, especially as the number of objective function increases. The BDDs performed particularly well for set packing problems, since they are typically much more compact when compared to the BDD representing MSCPs instances. The time to compile the BDDs was also less than 5 % of the total BDD time, as in the MKP case.

Table 2. Average solution times (in seconds) for MSCPs and MSPPs problems. Number in parentheses indicate instances unsolved within the time limit (out of 10).

		Set Covering (MSCP)			Set Packing (MSPP)		
n	p	$|\mathcal{Y}_N|$	Kirlik	BDD	$|\mathcal{Y}_N|$	Kirlik	BDD
100	3	117	5.91	5.46	164	5.61	5.07
	4	428	282.19	7.03	551.10	321.64	5.12
	5	1,171	$82.79^{(9)}$	7.87	1,211.90	(10)	5.35
150	3	305	23.99	110.50	336.00	12.81	8.37
	4	1,178	$1,228.18^{(2)}$	$248.44^{(2)}$	4,557	$3,025.52^{(8)}$	22.24
	5	4,711	(10)	$329.49^{(5)}$	9,213	(10)	49.18

6   Conclusions

This paper proposes an algorithm for solving general multiobjective discrete optimization problems using decision diagrams. We utilize decision diagrams to represent, exactly, the feasible set of the problem, and then uses a multicriteria shortest path algorithm for finding the set of nondominated solutions. The algorithm is applied to three classical discrete optimization problems, and computational methods indicate that the proposed method is superior to a state-of-the-art multiobjective technique, often providing orders of magnitude speedups.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Trans. Comput. **C−27**, 509–516 (1978)
2. Behle, M.: On threshold BDDs and the optimal variable ordering problem. J. Comb. Optim. **16**(2), 107–118 (2007)
3. Benson, H.P.: Existence of efficient solutions for vector maximization problems. J. Optim. Theory Appl. **26**(4), 569–580 (1978)
4. Bergman, D., Cire, A.A., van Hoeve, W.-J., Hooker, J.N.: Variable ordering for the application of BDDs to the maximum independent set problem. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 34–49. Springer, Heidelberg (2012)
5. Bergman, D., Cire, A.A., van Hoeve, W.-J., Hooker, J.N.: Discrete optimization with decision diagrams. INFORMS J. Comput. **28**(1), 47–66 (2016)
6. Bergman, D., Cire, A.A., Jan van Hoeve, W.-J., Yunes, T.H.: BDD-based heuristics for binary optimization. J. Heuristics **20**(2), 211–234 (2014)
7. Bergman, D., van Hoeve, W.-J., Hooker, J.N.: Manipulating MDD relaxations for combinatorial optimization. In: Achterberg, T., Beck, J.C. (eds.) CPAIOR 2011. LNCS, vol. 6697, pp. 20–35. Springer, Heidelberg (2011)
8. Bertsekas, D.P.: Dynamic Programming and Optimal Control, 2nd edn. Athena Scientific, Belmont (2000)
9. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. **C−35**, 677–691 (1986)
10. Bryant, R.E.: Symbolic boolean manipulation with ordered binary decision diagrams. ACM Comput. Surv. **24**, 293–318 (1992)
11. Chankong, V., Haines, Y.Y.: Multiobjective Decision Making: Theory and Methodology. Elsevier Science, New York (1983)
12. Chinchuluun, A., Pardalos, P.M.: A survey of recent developments in multiobjective optimization. Ann. Oper. Res. **154**(1), 29–50 (2007)
13. Coello, C.A.: An updated survey of GA-based multiobjective optimization techniques. ACM Comput. Surv. **32**(2), 109–143 (2000)
14. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons Inc., New York (2001)
15. Ehrgott, M.: A discussion of scalarization techniques for multiple objective integer programming. Ann. Oper. Res. **147**(1), 343–360 (2006)
16. Florios, K., Mavrotas, G.: Generation of the exact pareto set in multi-objective traveling salesman and set covering problems. Appl. Math. Comput. **237**, 1–19 (2014)
17. Garroppo, R.G., Giordano, S., Tavanti, L.: A survey on multi-constrained optimal path computation: exact and approximate algorithms. Comput. Netw. **54**(17), 3081–3107 (2010)
18. Haimes, Y.Y., Lasdon, L.S., Wismer, D.A.: On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE Trans. Syst. Man Cybern. **1**(3), 296–297 (1971)
19. Hansen, P.: Bicriterion path problems. In: Fandel, G., Gal, T. (eds.) Multiple Criteria Decision Making Theory and Application. Lecture Notes in Economics and Mathematical Systems, vol. 177, pp. 109–127. Springer, Heidelberg (1980)
20. Hooker, J.N.: Decision diagrams and dynamic programming. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013. LNCS, vol. 7874, pp. 94–110. Springer, Heidelberg (2013)

21. Hwang, C.-L., Masud, A.S.M.: Multiple Objective Decision Making Methods and Applications: A State-of-the-art Survey. Lecture Notes in Economics and Mathematical Systems, vol. 164. Springer, Heidelberg (1979)
22. IBM ILOG: Cplex optimization studio 12.6.3 user manual (2016)
23. Kirlik, G., Sayın, S.: A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. Eur. J. Oper. Res. **232**(3), 479–488 (2014)
24. Laumanns, M., Thiele, L., Zitzler, E.: An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. Eur. J. Oper. Res. **169**(3), 932–942 (2006)
25. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell Syst. Tech. J. **38**, 985–999 (1959)
26. Loui, R.P.: Optimal paths in graphs with stochastic or multidimensional weights. Commun. ACM **26**(9), 670–676 (1983)
27. Özlen, M., Azizoğlu, M.: Multi-objective integer programming: a general approach for generating all non-dominated solutions. Eur. J. Oper. Res. **199**(1), 25–35 (2009)
28. Miettinen, K.: Nonlinear Multiobjective Optimization. Kluwer Academic Publishers, Boston (1999)
29. Özlen, M., Burton, B.A., MacRae, C.A.G.: Multi-objective integer programming: an improved recursive algorithm. J. Optim. Theory Appl. **160**(2), 470–482 (2013)
30. Raith, A., Ehrgott, M.: A comparison of solution strategies for biobjective shortest path problems. Comput. Oper. Res. **36**(4), 1299–1331 (2009)
31. Figueira, J.R., Tavares, G., Wiecek, M.M.: Labeling algorithms for multiple objective integer knapsack problems. Comput. Oper. Res. **37**(4), 700–711 (2010)
32. Steuer, R.E., Choo, E.-U.: An interactive weighted Tchebycheff procedure for multiple objective programming. Math. Program. **26**(3), 326–344 (1983)
33. Stidsen, T., Andersen, K.A., Dammann, B.: A branch and bound algorithm for a class of biobjective mixed integer programs. Manage. Sci. **60**(4), 1009–1032 (2014)
34. Ulungu, E.L., Teghem, J.: Multi-objective combinatorial optimization problems: a survey. J. Multi-Criteria Decis. Anal. **3**(2), 83–104 (1994)
35. Vincke, P.: Problemes multicriteres. Cahiers Centre Etudes Recherche Operationnelle **16**, 425–439 (1974)
36. Wegener, I.: Branching programs and binary decision diagrams: theory and applications. In: SIAM Monograph Discrete Mathematics Applications. Society for Industrial and Applied Mathematics (2000)
37. Zhou, A., Qu, B.-Y., Li, H., Zhao, S.-Z., Suganthan, P.N., Zhang, Q.: Multiobjective evolutionary algorithms: a survey of the state of the art. Swarm Evol. Comput. **1**(1), 32–49 (2011)

Dependency Schemes in QBF Calculi: Semantics and Soundness

Olaf Beyersdorff and Joshua Blinkhorn[(✉)]

School of Computing, University of Leeds, Leeds, UK
{o.beyersdorff,scjlb}@leeds.ac.uk

Abstract. We study the parametrisation of QBF resolution calculi by dependency schemes. One of the main problems in this area is to understand for which dependency schemes the resulting calculi are sound. Towards this end we propose a semantic framework for variable independence based on 'exhibition' by QBF models, and use it to express a property of dependency schemes called *full exhibition* that is known to be sufficient for soundness in Q-resolution. Introducing a generalised form of the long-distance resolution rule, we propose a complete parametrisation of classical long-distance Q-resolution, and show that full exhibition remains sufficient for soundness. We demonstrate that our approach applies to the current research frontiers by proving that the reflexive resolution path dependency scheme is fully exhibited.

1 Introduction

The excellent success of SAT solvers in the realm of propositional Boolean formulae has motivated much interest in the corresponding search problem for quantified Boolean formulae (QBF). The greater expressiveness of QBF, afforded by its PSPACE-completeness [22], presents novel challenges in solving, and the array of emerging techniques is motivating a wealth of research in the closely-related field of proof complexity [3–8, 11–13, 24].

There is a natural correspondence between QBF practice and proof theory; when a solver concludes the falsity of an instance, the trace can be interpreted as a formal refutation. Understanding the refutational proof system that underpins a particular solving method, and thereby accounts for its correctness, motivates the proof-theoretic study of specific calculi. Recent work has led to a complete understanding of the relative strength of resolution-based QBF systems [3, 6], including Q-resolution (Q-Res) [14], universal Q-resolution (QU-Res) [24], and long-distance Q-resolution (LD-Q-Res) [1].

Implemented in the state-of-the-art solver DepQBF [15, 16], one of the recent and exciting developments in QBF solving has seen the introduction of *dependency schemes*: algorithms that gather information on variable independence by prior appeal to the syntactic form of an instance. The quantifier prefix of a QBF (in prenex normal form) imposes a total order on the variables; due to the nesting of quantifier scopes, the value of a Boolean variable z can be dependent upon the

variables to its left in the prefix. Naturally, this entails some restrictions on solving methods, and on the rules of the related formal systems. In general, however, z does not necessarily depend on all of the variables to its left. By identifiying *variable independence*, a dependency scheme attempts to replace the linear order of the prefix with a partial order, which more accurately reflects the dependency structure of the formula. This approach allows some sets of instances to be solved more effeciently, despite the compuational overhead incurred in computing the dependency scheme [15].

Independence itself is presented as a semantic concept [15,17]. The truth of a QBF Φ is witnessed by a Skolem-function model, a set of Boolean functions $\{f_x\}$ that produce a proposition tautology when substituted for the existential variables. The arguments to f_x are the universal variables U_x left of x in the quantifier prefix, but it may occur that some circuit computes f_x without using $u \in U_x$ as an input. In this case we say that x *is independent of* u – and a dual notion for false QBFs provides for independence of universals on existentials – even though the Skolem-function model is in general not unique.

This lack of uniqueness has consequences for soundness in QBF calculi. The impact of a dependency scheme in the proof system is to allow some logical steps which previously were prohibited; specifically, the $\forall$-reduction rule of Q-Res receives greater reign. This motivated the proposal of Q($\mathcal{D}$)-Res by Slivovsky and Szeider [21], a parametrisation of the classical calculus by dependency schemes. Some schemes that were previously put forward in the literature, such as the triangle [18] and resolution path [23] dependency schemes, have proved too aggressive for soundness in Q($\mathcal{D}$)-Res, admitting refutations of true QBFs. The reflexive resolution path dependency scheme [21] is currently the strongest known scheme for which Q($\mathcal{D}$)-Res is sound, a result which was proved by means of a difficult transformation of a Q($\mathcal{D}$)-Res refutation into a Q-Res refutation [21].

What is currently absent in the literature is a deeper understanding of soundness based on classification of dependency schemes; moreover, the lack of general methods may frustrate future developments. It is natural to propose the parametrisation by dependency schemes of stronger QBF calculi, of the other CDCL-based QBF resolution systems and QBF Frege [4], whereupon methods for proving soundness based on *properties* of dependency schemes will carry over. In this paper we demonstrate that semantic notions of independence are indeed equipped for this; our contributions are summarized below.

1. New QBF Calculi Parametrised by Dependency Schemes. We extend the parametrisation by dependency schemes to all the CDCL-based resolution calculi for QBF: with the new long-distance calculus LD-Q($\mathcal{D}$)-Res, with universal resolution QU($\mathcal{D}$)-Res, and with their combination LQU($\mathcal{D}$)-Res. Our new long-distance calculus presents the greatest challenge. Of the two inference rules employed classically, parametrisation of $\forall$-reduction can be lifted straight from Q($\mathcal{D}$)-Res; here we investigate the additional effects of parametrising the long-distance resolution rule as well, by relaxing the conditions under which so-called 'merged literals' can be introduced. Progressing from Q-resolution,

we demonstrate that variable independence and merging have a more subtle interaction; in LD-Q($\mathcal{D}$)-Res, we must supplant merged literals with *annotated literals*, which record existential pivots to prevent unsound $\forall$-reduction steps.

2. A Semantic Framework for Independence and Soundness. We unify some existing approaches in the literature towards a more fruitful understanding of the interplay between Q-resolution and dependency schemes. Building on the work of Samer [17] and Lonsing [15] we propose a semantic framework for variable independence. Central to the framework is a property of dependency schemes called *full exhibition*, which was shown to be sufficient for soundness in Q($\mathcal{D}$)-Res by Slivovsky [20]. We further the potential of this approach to show that full exhibition is sufficient for soundness in all the dependency calculi we introduce. To that end, we handle the semantic obstacles of long-distance resolution by incorporating techniques from strategy extraction due to Balabanov et al. [2].

3. Demonstrating Full Exhibition. We conclude by proving Slivovsky's conjecture [20, p. 37] that the reflexive resolution path dependency scheme $\mathcal{D}^{\mathrm{rrs}}$ is fully exhibited. Currently, $\mathcal{D}^{\mathrm{rrs}}$ is arguably the most important dependency scheme, capable of revealing more cases of independence than any other tractable scheme known to be sound for Q($\mathcal{D}$)-Res. As such, we show that everything currently known about soundness in this setting can be explained by full exhibition. On the technical level, the result is obtained by an algorithmic transformation of an arbitrary model for a true QBF Φ into a model that exhibits all the required independencies. We therefore reveal the possibility for QBF solving to implement long-distance techniques fully parametrised by $\mathcal{D}^{\mathrm{rrs}}$, or any other fully exhibited scheme.

Organisation of the Paper. After providing the necessary fundamentals in Sect. 2, we present our semantic framework based on 'exhibition' in Sect. 3. In Sect. 4, we present the new long-distance calculus and corresponding soundness results, while Sect. 5 covers the proof that $\mathcal{D}^{\mathrm{rrs}}$ is fully-exhibited. Finally, some conclusions are offered in Sect. 6.

2   Preliminaries

Quantified Boolean Formulas. A *Quantified Boolean Formula* (QBF) Φ over a set $V = \{z_1, \ldots, z_n\}$ of n variables is a formula in quantified Boolean logic with variables ranging over $\{0, 1\}$. We consider only formulas in *prenex conjunctive normal form* (PCNF), denoted $\Phi = \mathcal{Q} . \phi$, in which all variables are quantified either existentially or universally in the *quantifier prefix* $\mathcal{Q} = \mathcal{Q}_1 z_1 \cdots \mathcal{Q}_n z_n$, $\mathcal{Q}_i \in \{\exists, \forall\}$ for $i \in [n]$, and ϕ is a propositional conjunctive normal form (CNF) formula called the *matrix*. A CNF matrix is a conjunction of clauses, each clause is a disjunction of literals, and a literal is a variable or its negation. Whenever convenient, we refer to a clause as a set of literals and to a matrix as a set of
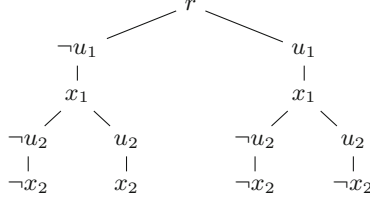
Fig. 1. An assignment tree T for a PCNF $\forall u_1 \exists x_1 \forall u_2 \exists x_2 . \phi$, with arbitrary matrix ϕ.

clauses. We typically write x for existential variables, u and v for universals, and z for either. We denote the sets of existentially and universally quantified variables of Φ by $V_\exists = \{z_i \in V \mid \mathcal{Q}_i = \exists\}$ and $V_\forall = \{z_i \in V \mid \mathcal{Q}_i = \forall\}$ respectively. The prefix $\mathcal{Q}$ imposes a linear ordering $<_\Phi$ on the variables of Φ, such that $z_i <_\Phi z_j$ holds whenever $i < j$, in which case we say that z_j *is right of* z_i, or that z_i *is left of* z_j. The sets of variables right and left of z are denoted $R_\Phi(z) = \{z' \in V \mid z <_\Phi z'\}$ and $L_\Phi(z) = \{z' \in V \mid z' <_\Phi z\}$.

Assignment Trees and Models. Assignment trees for PCNFs were first introduced in [19]. We represent an assignment tree formally as a set of paths. Let Φ be a PCNF over variables $V = \{z_1, \dots, z_n\}$ and let $V_\forall = \{u_1, \dots, u_k\}$. A path is a set of literals $P = \{l_1, \dots, l_n\}$ with $\mathrm{var}(l_i) = z_i$ for all $i \in [n]$, and we write $P[z_i] = l_i$. A set of paths T is well-formed for Φ iff (1) for all $u \in V_\forall$ and for all $P, Q \in T$, if $P[v] = Q[v]$ for all $v \in L_\Phi(u) \cap V_\forall$, then $P[x] = Q[x]$ for each $x \in L_\Phi(u) \cap V_\exists$, and (2) for each set of literals $U = \{l_1, \dots, l_k\}$ with $\mathrm{var}(l_i) = u_i$ for $i \in [k]$, there is a unique path $P \in T$ with $U \subseteq P$. A set of paths that is well-formed for Φ is an *assignment tree* for Φ. We also use P to denote the total assignment $P : V \to \{\top, \bot\}$ given by $P(z_i) = \bot$ if $l_i = \neg z_i$ and $P(z_i) = \top$ if $l_i = z_i$, and extend this notation to literals with $P(\neg z_i) = \neg P(z_i)$, where $\top = \neg\bot$ and vice versa. An assignment tree for Φ is a *model* for Φ, typically denoted M, iff $P(C) = \top$ for all paths $P \in T$ and all clauses $C \in \phi$, where $P(C) = \top$ iff $P(l) = \top$ for some $l \in C$. A PCNF which has a model is *true*, otherwise it is *false*. An assignment tree is depicted as a tree with root r, as shown in Fig. 1.

Dependency Schemes. The trivial dependency scheme $\mathcal{D}^{\mathrm{trv}}$ is a mapping which associates each PCNF $\Phi = \mathcal{Q}_1 z_1 \cdots \mathcal{Q}_n z_n . \phi$ over variables V to the trivial dependency relation $\mathcal{D}_\Phi^{\mathrm{trv}} = \{(z_i, z_j) \mid i < j \text{ and } \mathcal{Q}_i \neq \mathcal{Q}_j\}$. A *proto-dependency scheme*[1] $\mathcal{D}$ is a function that maps each PCNF Φ to a binary relation $\mathcal{D}_\Phi \subseteq \mathcal{D}_\Phi^{\mathrm{trv}}$ called the *dependency relation*. If $(z_i, z_j) \in \mathcal{D}_\Phi$, then (z_i, z_j) is a $\mathcal{D}$-dependency

---

[1] The term 'dependency scheme' was first introduced to denote a subset of proto-dependency schemes with a more technical definition [18]; for consistency with the literature we will use 'proto-dependency scheme' in technical portions of this paper.

and z_j is a $\mathcal{D}$-dependent of z_i, otherwise z_j is $\mathcal{D}$-independent of z_i. A proto-dependency scheme $\mathcal{D}'$ is said to be *at least as general* as another $\mathcal{D}$ if $\mathcal{D}'_\Phi \subseteq \mathcal{D}_\Phi$ for all PCNFs Φ, and is *strictly more general* if the inclusion is strict for some formula. For a PCNF Φ over variables V and $u \in V_\forall$, we write $\bar{\mathcal{D}}_\Phi(u) = \{(u, x) \mid x \in V_\exists \text{ and } (u, x) \notin \mathcal{D}_\Phi\}$.

QBF Resolution Calculi. We give a brief overview of four resolution-based CDCL QBF calculi – see [6] for a more detailed survey. Their formal definitions are presented in Sects. 3 and 4 as special cases of the corresponding 'dependency' systems. A refutational QBF calculus is *sound* iff the empty clause cannot be derived from any true formula.

Q-resolution (Q-Res) introduced in [14] is the standard refutational calculus for PCNF. In addition to resolution over existential pivots with non-tautologous resolvents, the calculus has a universal reduction rule which allows a clause C to be derived from $C \cup \{u\}$, where u is a universal literal and all existential literals in C are left of u. *QU-resolution* (QU-Res) [24] is a natural extension of Q-Res that allows universal resolution pivots.

Long-distance resolution, which was introduced in [25] and formalised as the calculus LD-Q-Res [1], allows tautologous resolvents under certain conditions, using the special merged literal u^* to represent the tautology $\{u, \neg u\}$. The resulting system is exponentially stronger than Q-Res [12]. Finally, the calculus LQU-Res [3] combines naturally the features of QU-Res and LD-Q-Res, allowing merged literals and resolution over universal pivots.

3   Dependency Schemes, Q-resolution and Semantics

3.1   Dependency Schemes and Q-resolution

It is natural to try to strengthen a classical QBF calculus using a dependency scheme, and the starting point for the 'dependency version' is the identification of the trivial dependency relation in the rules of a calculus. Restrictions are inevitably imposed by the linear ordering of the quantifier prefix. Dependency calculi can relax these restrictions, replacing the implicit reference to $\mathcal{D}^{\mathrm{trv}}$ with an explicit reference to a more general dependency scheme $\mathcal{D}$.

Figure 2 recalls the rules of Q($\mathcal{D}$)-Res, the dependency version of Q-Res, introduced in [21] to account for the behaviour of the QDPLL-based solver DepQBF [9,15]. In Q-Res, the universal reduction rule allows a universal u to be dropped from a clause C containing only existential variables left of u. By comparison, Q($\mathcal{D}$)-Res allows u to be dropped whenever C contains no $\mathcal{D}$-dependents of u. Note that Q-Res and Q($\mathcal{D}^{\mathrm{trv}}$)-Res are identical. Whether or not Q($\mathcal{D}$)-Res is sound depends on the strength of the dependency scheme. For example, in [21] it is shown that Q($\mathcal{D}$)-Res is sound for $\mathcal{D}^{\mathrm{rrs}}$, but unsound for the strictly more general scheme $\mathcal{D}^{\mathrm{res}}$.

It is natural to extend Q($\mathcal{D}$)-Res by allowing resolution over universal pivots. The resulting new system QU($\mathcal{D}$)-Res, also presented in Fig. 2, is the dependency version of QU-Res.

$$\frac{}{C} \ (\text{Axiom}) \qquad\qquad C \text{ is a clause in the matrix of } \Phi.$$

$$\frac{D \cup \{l_u\}}{D} \ (\forall\text{-Red}) \qquad \begin{array}{l} \text{Literal } l_u \text{ is universal. If } l \in D \text{ and} \\ \text{var}(l) = v, \text{ then } (u, v) \notin \mathcal{D}_\Phi. \end{array}$$

$$\frac{C_1 \cup \{v\} \qquad C_2 \cup \{\neg v\}}{C_1 \cup C_2} \ (\text{Res}) \qquad \begin{array}{l} \text{If } l \in C_1, \text{ then } \neg l \notin C_2. \text{ In Q}(\mathcal{D})\text{-Res,} \\ \text{variable } v \text{ is existential; in QU}(\mathcal{D})\text{-} \\ \text{Res, } v \text{ is existential or universal.} \end{array}$$

Fig. 2. The rules of Q($\mathcal{D}$)-Res [21] and QU($\mathcal{D}$)-Res

3.2 A Semantic Framework for Independence

We reformulate the definition of independence in terms of assignment trees from [15,17]; we feel our notation is better suited to the aims of the current work. We first introduce the new idea of *complementary paths* in an assignment tree, whose universal literals differ for exactly one variable.

Definition 1 (Complementary path). *Let Φ be a QBF over variables V, let U be a non-tautologous set of literals such that $\text{var}(U) = V_\forall$, let T be an assignment tree for Φ and let $P \in T$ be the unique path such that $U \subseteq P$. Then, for any $u \in V_\forall$, $P_u \in T$ is the unique path such that $U' \subseteq P_u$, where $U' = (U \setminus \{l\}) \cup \{\neg l\}$, $l \in U$ and $\text{var}(l) = u$.*

It is fortunate that, throughout this paper, we need only consider the dependence of existentials on universals. This simplification, seen in the definition below, is the result of our dealing exclusively with refutational calculi, the rules of which remain unaffected by the (in)dependence of universals on existentials.

Definition 2 (Independence of existentials from universals [15,17]). *Let Φ be a true QBF over variables V and let $u \in V_\forall$, $x \in V_\exists$. We say that x is independent of u in Φ if there exists a model M for Φ in which $P(x) = P_u(x)$ for all paths $P \in M$. For such a model M we write $M \prec (u, x)$, and we say that M exhibits the independence of x from u in Φ.*

Remark 3. It is not necessary for us to consider false QBFs in Definition 2, since, by definition, a false formula has no models. For a false formula, the condition for independence is satisfied vacuously, confirming our intuition that no existential variable can be dependent on any universal in such a formula.

As noted in [21], Definition 2 alone is too weak for soundness in Q($\mathcal{D}$)-Res. The problem lies in the possibility for different models to exhibit different independencies, which are then used together in the same refutation. It is therefore natural to seek a model which exhibits all the independencies that may be used in a refutation.

Definition 4 (Fully exhibited dependency scheme). *Let $\mathcal{D}$ be a proto-dependency scheme. We say that $\mathcal{D}$ is* fully exhibited *iff for each true PCNF Φ there is a model M for Φ such that $M \prec (u, x)$ for each pair $(u, x) \notin D_\Phi$, with $u \in V_\forall$ and $x \in V_\exists$.*

In [20], it was proved that $\mathsf{Q}(\mathcal{D})$-Res is sound for fully exhibited[2] $\mathcal{D}$, and this was combined with the fact that the standard dependency scheme $\mathcal{D}^{\mathrm{std}}$ is fully exhibited (attributed to [10]). In the next section, we show that this approach scales up to the dependency versions of stronger QBF calculi.

4   Dependency Schemes and Long-Distance Q-resolution

In this section, we introduce the new long-distance calculi $\mathsf{LD}\text{-}\mathsf{Q}(\mathcal{D})$-Res and $\mathsf{LQU}(\mathcal{D})$-Res, the respective dependency versions of $\mathsf{LD}\text{-}\mathsf{Q}$-Res and LQU-Res.

Long-distance Q-resolution was formalised as a calculus [1] to account for solving techniques due to [25]. The resulting system is exponentially stronger than Q-Res [12]. The salient feature of the system is that tautological clauses are allowed under certain conditions. Specifically, resolving clauses C_1 and C_2 over an existential pivot x, a 'merged literal' u^* appears in the resolvent clause C if $\neg u \in C_1$, $u \in C_2$ and $x <_\Phi u$. In successive resolution steps, a merged literal u^* may be merged again with another merged literal u^*, or with non-merged literals u and $\neg u$, provided that the existential pivot is left of u. Both merged and non-merged literals may be dropped from a clause by $\forall$-reduction under the usual conditions.

Whereas the parametrisation of $\forall$-reduction can be lifted directly from $\mathsf{Q}(\mathcal{D})$-Res, parametrisation of long-distance resolution, which relaxes the conditions under which merging is allowed, presents a novel challenge.

4.1   Defining $\mathsf{LD}\text{-}\mathsf{Q}(\mathcal{D})$-Res and $\mathsf{LQU}(\mathcal{D})$-Res

Although the method of generalising the reference to the trivial dependency scheme remains, more care must be taken when defining $\mathsf{LD}\text{-}\mathsf{Q}(\mathcal{D})$-Res. Parametrising long-distance resolution means relaxing the conditions under which merging may take place, which in turn entails some new notation. Replacing $x <_\Phi u$ with the condition $(u, x) \notin \mathcal{D}_\Phi$ in the dependency version, one must annotate merged literals with the corresponding *pivot set X*, producing an *annotated literal $u^X \in C$*, where X consists of all the existential variables over which u has been merged in the derivation of the clause C. Annotations are needed to keep track of the pivot sets to prevent unsound $\forall$-reduction steps – we explain this in greater detail shortly.

The rules of $\mathsf{LD}\text{-}\mathsf{Q}(\mathcal{D})$-Res are given in Fig. 3. We observe that $\mathsf{LD}\text{-}\mathsf{Q}(\mathcal{D}^{\mathrm{trv}})$-Res is precisely the classical long-distance calculus $\mathsf{LD}\text{-}\mathsf{Q}$-Res, except that the merged literals of the latter are annotated. Since the dependency conditions of $\mathsf{LD}\text{-}\mathsf{Q}(\mathcal{D})$-Res are identical to the classical long-distance conditions if $\mathcal{D}$ is

---

[2] Full exhibition is treated equivalently, as a property of models.

$\mathcal{D}^{\mathrm{trv}}$, replacing all annotated literals u^X in an LD-Q($\mathcal{D}^{\mathrm{trv}}$)-Res refutation with merged literals u^* produces an LD-Q-Res refutation, and vice versa – replacing all merged literals u^* in an LD-Q-Res refutation with annotated literals u^X produces an LD-Q($\mathcal{D}^{\mathrm{trv}}$)-Res refutation. Similarly as for Q($\mathcal{D}$)-Res, it is natural to extend LD-Q($\mathcal{D}$)-Res by allowing resolution over universal pivots. The resulting new system LQU($\mathcal{D}$)-Res, also given in Fig. 3, is the dependency version of LQU-Res.

$$\frac{}{C} \; \text{(Axiom)} \qquad\qquad C \text{ is a clause in the matrix of } \Phi.$$

$$\frac{D \cup \{u^X\}}{D} \; (\forall\text{-Red})$$

Variable u is universal. If $l \in D$ and $\mathrm{var}(l) = z$, then $(u, z) \notin \mathcal{D}_\Phi$, and if $l = z^{X'}$ then $(u, x) \notin \mathcal{D}_\Phi$ for all $x \in X'$. If $X = \emptyset$ then literal u^X is either u or $\neg u$.

$$\frac{C_1 \cup U_1 \cup \{x\} \qquad C_2 \cup U_2 \cup \{\neg x\}}{C_1 \cup C_2 \cup U} \; \text{(Res)}$$

If for $l_1 \in C_1, l_2 \in C_2, \mathrm{var}(l_1) = \mathrm{var}(l_2)$, then $l_1 = l_2$ is not annotated. $\mathrm{var}(U_1) = \mathrm{var}(U_2) \subseteq V_\forall$, and $(x, u) \notin \mathcal{D}_\Phi$ for each $u \in \mathrm{var}(U_1)$. If for $u_1 \in U_1, u_2 \in U_2, \mathrm{var}(u_1) = \mathrm{var}(u_2) = u$, then $u_1 = \neg u_2$, or at least one of u_1, u_2 is annotated. U is defined as $\{u^X \mid u \in \mathrm{var}(U_1)\}$, where X is the union of $\{x\}$ with any annotations on u in $U_1 \cup U_2$. In LD-Q($\mathcal{D}$)-Res $\mathrm{var}(x)$ is existential. In LQU($\mathcal{D}$)-Res, $\mathrm{var}(x)$ is existential or universal.

Fig. 3. The rules of LD-Q($\mathcal{D}$)-Res

The purpose of annotating literals is to prevent unsound $\forall$-reduction steps, by checking that the pivot sets in the clause are $\mathcal{D}$-independent of the reduced universal variable. Annotations were never necessary in LD-Q-Res; the fact that a merged literal u^* in the clause is always right of its corresponding existential pivots is enough to ensure soundness. However, in LD-Q($\mathcal{D}$)-Res, we must explicitly forbid $\forall$-reduction of $v \in C$ if any $x \in X$ is not $\mathcal{D}$-independent of v, for any annotation X in the clause C. The following example shows that allowing v to be reduced under such conditions is unsound in general for a fully-exhibited proto-dependency scheme $\mathcal{D}$.

Example 5. Take the true QBF $\Psi = \forall u \exists x_1 \forall v \exists x_2 \exists x_3 . \phi$ with the matrix $\phi = \big\{\{u, x_2, \neg x_3\}, \{\neg u, \neg x_2, \neg x_3\}, \{x_1, v, x_3\}, \{\neg x_1, \neg v, x_3\}\big\}$ and the proto-dependency scheme $\mathcal{D}'_\Phi = \{(u, x_1), (v, x_2), (u, x_3), (v, x_3)\}$ if $\Phi = \Psi$, and $\mathcal{D}'_\Phi = \mathcal{D}^{\mathrm{trv}}_\Phi$ otherwise. First observe that $\mathcal{D}'$ is fully exhibited; Fig. 4 depicts a model M for Ψ which exhibits the independence of x_2 on u.
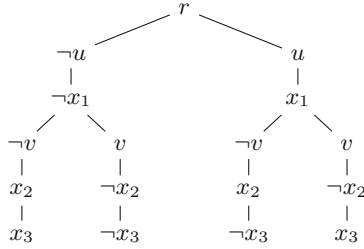
Fig. 4. A model M for Ψ for which $M \prec (u, x_2)$.

However, if we allow variable v to be reduced alongside the annotated literal $u^{\{x_2\}}$, noting that x_2 is not $\mathcal{D}'$-independent of v, we obtain the following refutation of Ψ.

$$\frac{\dfrac{\{u, x_2, \neg x_3\} \qquad \{\neg u, \neg x_2, \neg x_3\}}{\{u^{\{x_2\}}, \neg x_3\}} \qquad \dfrac{\{x_1, v, x_3\} \qquad \{\neg x_1, \neg v, x_3\}}{\{v^{\{x_1\}}, x_3\}}}{\dfrac{\dfrac{\{u^{\{x_2\}}, v^{\{x_1\}}\}}{\{u^{\{x_2\}}\}}}{\bot}}$$

4.2   Soundness of **LD-Q($\mathcal{D}$)-Res** and **LQU($\mathcal{D}$)-Res**

In this subsection, we prove that LD-Q($\mathcal{D}$)-Res is sound for a fully exhibited $\mathcal{D}$, and our method entails the following evaluation of annotated literals under assignment. We define *annotated literal functions*, which are based on the 'phase functions' and 'effective literals' introduced by Balabanov et al. [2].

Informally, it is demonstrated in [2] that any assignment σ to the existential variables 'induces' the phase of a merged literal u^* in an LD-Q-Res refutation of a PCNF Φ, such that for the purpose of strategy extraction it may be interpreted as either non-merged literal u or $\neg u$. In a given model M for Φ, every path P contains a particular assignment to the existential variables. Therefore, for any annotated literal u^X in some LD-Q($\mathcal{D}$)-Res derivation from Φ, we can use the phase function to associate a non-annotated literal u or $\neg u$ with P. This allow us to evaluate annotated literals, and the nature of the phase function ensures that the rules of LD-Q($\mathcal{D}$)-Res are logically correct for each path in a fully exhibiting model. For a given annotated literal, our annotated literal function uses the same method as Balabanov et al. to identify the correct phase induced by some existential assignment. However, since we are not concerned with strategy extraction, we are able to simplify the construction considerably compared to [2][3]. For that reason, we proceed as follows.

---

[3] We can prove what we need to from the definition of such functions; we need not represent them explicitly as circuits as in [2].

As a starting point, consider the following resolution step in an LD-Q-Res refutation of a QBF $\Phi = \mathcal{Q} . \phi$ over variables V, where $\text{var}(l_u) = u \in V_\forall$ and $x \in V_\exists$.

$$\frac{C_1 \cup \{x\} \cup \{l_u\} \qquad C_2 \cup \{\neg x\} \cup \{\neg l_u\}}{C_1 \cup C_2 \cup \{u^*\}}$$

Let M be any model for the conjunction of the antecedent clauses prefixed by $\mathcal{Q}$, and let $P \in M$. For any universal v which is right of u, we must have $x <_\Phi u <_\Phi v$; therefore $P(x) = P_v(x)$, meaning that at least one of $C_1 \cup \{u\}$ and $C_2 \cup \{\neg u\}$ is satisfied by both P and P_v. In either case we can then choose a single literal u or $\neg u$ for u^* such that $C_1 \cup C_2 \cup \{u^*\}$ is satisfied by both P and P_v.

Generalising, let M be a model for Φ. We observe that any resolution step producing u^* from complementary literals gives rise to a well-defined function $f_u^* : M \to \{u, \neg u\}$, with rule

$$f_u^*(P) = \begin{cases} l_u & \text{if } P(x) = \bot \, , \\ \neg l_u & \text{if } P(x) = \top \, . \end{cases}$$

We observe two features of such a definition. First, f_u^* simply reads the truth value of $P(x)$, selects the antecedent clause in which the pivot variable x is falsified, and takes the universal literal from that clause. In this way, any path $P \in M$ made to satisfy $C_1 \cup C_2 \cup f_u^*(P)$. Second, if v is any universal right of u, then $P(x) = P_v(x)$; hence v satisfies the *complementary property* $f_u^*(P) = f_u^*(P_v)$ for all $P \in M$.

Moreover, the above discussion does not consider 'successive merging'. Moving forward to the annotated literals of LD-Q($\mathcal{D}$)-Res, we therefore present a recursive definition based on the preceeding discussion.

Definition 6 (Annotated literal function). *Let $u^X \in U$ be a literal introduced by merging universal literals $l_1 \in U_1$ and $l_2 \in U_2$ in a resolution step*

$$\frac{C_1 \cup U_1 \cup \{x\} \qquad C_2 \cup U_2 \cup \{\neg x\}}{C_1 \cup C_2 \cup U}$$

of an LD-Q($\mathcal{D}$)-Res refutation of a formula Φ. Let X_1, X_2 be the resolution sets of l_1, l_2 respectively, and let M be a model for Φ. Then the annotated literal function $f_u^X : M \to \{u, \neg u\}$ for M is given by

$$f_u^X(P) = \begin{cases} l_1 & \text{if } P(x) = \bot \text{ and } X_1 = \emptyset \, , \\ f_u^{X_1}(P) & \text{if } P(x) = \bot \text{ and } X_1 \neq \emptyset \, , \\ l_2 & \text{if } P(x) = \top \text{ and } X_2 = \emptyset \, , \\ f_u^{X_2}(P) & \text{if } P(x) = \top \text{ and } X_2 \neq \emptyset \, , \end{cases}$$

where $X = X_1 \cup X_2 \cup \{x\}$.

The following lemma states that the complementary property holds for annotated literal functions.

Lemma 7. *Let Φ be a QBF over variables V, let $u, v \in V_\forall$, let $X \subseteq V_\exists$ and let M be a model for Φ for which $M \prec (v, x)$ for all $x \in X$. Then any annotated literal function f_u^X for M satisfies $f_u^X(P) = f_u^X(P_v)$ for all paths $P \in T$.*

Proof. The lemma follows from the observation that, throughout the recursive definition of the annotated literal function f_u^X, complementary paths P and P_v always map to the same case, since $P(x) = P_v(x)$ for all $x \in X$ and for all $P \in M$. ∎

Evaluation of Annotated Literals. We defined annotated literal functions for a model M specifically so that any $P \in M$ satisfying both antecedents of a resolution step also satisfies the resolvent. For that reason, we define u^X to have the same truth value as the concrete literal $f_u^X(P)$ when evaluated under α, the assignment represented by path P; that is, we define $(u^X)|_\alpha = (f_u^X(P))|_\alpha$. Representing assignments by paths, this would be written $P(u^X) = P(f_u^X(P))$. The expression $P(f_u^X(P))$ is always well-defined because f_u^X can be computed for any given model M, so $f_u^X(P)$ is a well-defined non-annotated literal, which can then be evaluated under P in the usual way. We are now in a position to prove the following theorem.

Theorem 8. *Let $\mathcal{D}$ be a fully exhibited proto-dependency scheme. Then $\mathsf{LD\text{-}Q(\mathcal{D})\text{-}Res}$ is sound.*

Proof. Let $\Phi = \mathcal{Q}.\phi$ be a QBF over variables V, suppose that $\pi = \{C_1, \ldots, C_l\}$ is a $\mathsf{LD\text{-}Q(\mathcal{D})\text{-}Res}$ refutation of Φ, and let

$$\phi_i = \begin{cases} \phi & \text{if } i = 0, \\ \phi \wedge C_1 \wedge \cdots \wedge C_i & \text{otherwise}, \end{cases}$$

for $i = 1, \ldots, l$. Since $\mathcal{D}$ is fully exhibited, if Φ is true there exists a model M for Φ for which $M \prec \bar{\mathcal{D}}_\Phi^{\mathrm{rrs}}(u)$ for all $u \in V_\exists$. We prove by induction on i that if Φ is true, M is a model for $\mathcal{Q}.\phi_i$, so $\mathcal{Q}.\phi_i$ is true. Hence at step $i = l$, we deduce that $\Phi = \mathcal{Q}.\phi_l$ is true, a clear contradiction since ϕ_l contains the empty clause C_l. Since $\mathcal{Q}.\phi = \mathcal{Q}.\phi_0$, if Φ is true then M is a model for $\mathcal{Q}.\phi_0$, thus the base case $i = 0$ is established. We only need confirm that if M is a model for $\mathcal{Q}.\phi_i$, then M is a model for $\mathcal{Q}.\phi_{i+1}$, for $i \in [l-1]$.

Suppose that $C_{i+1} = C_1 \cup C_2 \cup U$ is the resolvent of clauses $C_j = C_1 \cup U_1 \cup \{x\}$ and $C_k = C_2 \cup U_2 \cup \{\neg x\}$ for $j, k < i + 1$, and let P be an arbitrary path in M. By the inductive hypothesis, P satisfies C_j and C_k. Assume without loss of generality that $P(x) = \bot$. Then P satisfies $C_1 \cup U_1$. If P satisfies C_1 then P satisfies C_{i+1}. Otherwise, $P(l_u) = \top$ for some (annotated or non-annotated) literal $l_u \in U_1$ with $\mathrm{var}(l_u) = u$. The recursive definition of the annotated literal function ensures that $P(l_u) = \top \Rightarrow P(u^X) = \top$ for some annotated literal $u^X \in U$, and so P satisfies C_{i+1}. Therefore M is a model for $\mathcal{Q}.\phi_{i+1}$.

On the other hand, suppose that C_{i+1} was obtained from $C_j, j < i + 1$, by $\forall$-reduction on a non-annotated universal literal u. Then $C_{i+1} = C_j \backslash \{l_u\}$, where $\mathrm{var}(l_u) = u$, $(u, x) \notin \mathcal{D}_\Phi$ for all $x \in C_j \cap V_\exists$ and for all $x \in X$, where X is the union of the resolution sets of all universal literals in C_j. Suppose that there exists some path P in M which satisfies C_j but falsifies C_{i+1}. Let $z \in C_{i+1}$; then $P(z) = \bot$, and since $M \prec_\Phi (u, x)$ for all $(u, x) \notin \mathcal{D}_\Phi$, we have $P_u(z) = P(z) = \bot$ whenever z is a non-annotated literal. On the other hand, suppose that $z = v^X$, where $v \neq u$; then, since $M \prec_\Phi (u, x)$ for all $x \in X$, Lemma 7 gives $f_v^X(P) = f_v^X(P_u) = l_v$ with $\mathrm{var}(l_v) = v$, which implies $P^u(v^X) = P(v^X) = \bot$. Also, since $P(u) = \top$, we have $P_u(u) = \bot$, and we deduce that $P_u(C_j) = \bot$, contradicting that M is a model for $\mathcal{Q}.\phi_i$. It follows that P satisfies C_{i+1}, and that M is a model for $\mathcal{Q}.\phi_{i+1}$.

The same argument applies to an annotated literal u^X. Since $M \prec_\Phi (u, x)$ for all $x \in X$, the special case of Lemma 7 with $v = u$ gives $f_u^X(P) = f_u^X(P_u)$, hence $P(u^X) = \top$ implies $P_u(u^X) = \bot$. This completes the proof.   □

Since the proof of Theorem 8 makes no use of the fact that the pivot is existential, it also shows the soundness of $\mathsf{LQU}(\mathcal{D})$-$\mathsf{Res}$, the 'dependency version' of LQU-Res, for any fully exhibited $\mathcal{D}$.

Theorem 9. *Let $\mathcal{D}$ be a fully exhibited proto-dependency scheme. Then $\mathsf{LQU}(\mathcal{D})$-$\mathsf{Res}$ is sound.*

Also, since $\mathsf{LQU}(\mathcal{D})$-$\mathsf{Res}$ clearly simulates $\mathsf{QU}(\mathcal{D})$-Res simply by disallowing long-distance resolution steps, we obtain same result for $\mathsf{QU}(\mathcal{D})$-Res.

Theorem 10. *Let $\mathcal{D}$ be a fully exhibited proto-dependency scheme. Then $\mathsf{QU}(\mathcal{D})$-Res is sound.*

Theorems 8, 9 and 10 together constitute the generalisation to all the CDCL QBF calculi of Slivovsky's result [20] that $\mathsf{Q}(\mathcal{D})$-Res is sound for fully exhibited $\mathcal{D}$. Whereas full exhibition is a sufficient condition for each calculus, it is not a necessary condition for any of them, witnessed by the following counter example.

Example 11. Consider the formula $\Psi = \forall u_1 \forall u_2 \exists x_1 \exists x_2 . \psi$ with matrix

$$\psi = \big\{\{u_1, x_1, \neg x_2\}_1, \{u_1, \neg x_1, x_2\}_2, \{\neg u_1, u_2, x_1, \neg x_2\}_3, \{\neg u_1, u_2, \neg x_1, x_2\}_4,$$
$$\{\neg u_1, \neg u_2, x_1, x_2\}_5, \{\neg u_1, \neg u_2, \neg x_1, \neg x_2\}_6\big\},$$

and the dependency scheme $\mathcal{D}'$ defined by $\mathcal{D}'(\Phi) = \{(u_1, x_1), (u_2, x_2)\}$ if $\Phi = \Psi$ and $\mathcal{D}'(\Phi) = \mathcal{D}^{\mathrm{trv}}(\Phi)$ otherwise. It can be verified that Ψ is true, but there is no model for Ψ which exhibits both independencies (u_1, x_2) and (u_2, x_1) simultaneously, and hence $\mathcal{D}'$ is not fully exhibited. However, there is no $\mathsf{LQU}(\mathcal{D}')$-$\mathsf{Res}$ refutation of Ψ. One may resolve clauses 1 and 3 over u_1 to obtain $\{u_2, x_1, \neg x_2\}$, and resolve over clauses 2 and 4 to obtain $\{u_2, \neg x_1, x_2\}$. Beyond these two steps, no more $\mathsf{LQU}(\mathcal{D})$-$\mathsf{Res}$ steps can be made.

5   Demonstrating Full Exhibition

In this section, we demonstrate that the reflexive resolution path dependency scheme $\mathcal{D}^{\mathrm{rrs}}$ [21] is fully exhibited, thereby proving the conjecture of Slivovsky [20, p. 37]. This result provides a better understanding of soundness in Q-resolution with dependency shemes; since $\mathcal{D}^{\mathrm{rrs}}$ is the most general scheme known to be sound in $\mathsf{Q}(\mathcal{D})$-Res, what is already known about soundness for that calculus can subsequently be explained entirely by full exhibition.

The scheme $\mathcal{D}^{\mathrm{rrs}}$ uses the notion of 'resolution paths' introduced in [23], which define connections through the matrix with respect to a particular set of variables. For convenience, we represent the connections used in $\mathcal{D}^{\mathrm{rrs}}$ as a binary relation $\mathcal{C}_\Phi$.

Definition 12. *Let $\Phi = \mathcal{Q} \, . \, \phi$ be a PCNF over variables V and let l, l' be literals such that $(\mathrm{var}(l), \mathrm{var}(l')) \in \mathcal{D}^{\mathrm{trv}}_\Phi$. Then $(l, l') \in \mathcal{C}_\Phi$ iff there is a sequence of clauses $C_1, \ldots, C_n \in \phi$ and a sequence of literals $l_1, \ldots, l_{n-1} \in V_\exists \cap \mathcal{R}_\Phi(\mathrm{var}(l))$ such that $l_i \in C_i$, $\neg l_i \in C_{i+1}$ and $\mathrm{var}(l_i) \neq \mathrm{var}(l_{i+1})$ for $i \in [n-1]$.*

Definition 13 (Reflexive resolution path dependency scheme [21]). *The reflexive resolution path dependency scheme $\mathcal{D}^{\mathrm{rrs}}$ maps each PCNF Φ to the dependency relation*

$$\mathcal{D}^{\mathrm{rrs}}_\Phi = \{(z_1, z_2) \in \mathcal{D}^{\mathrm{trv}}_\Phi \mid (z_1, z_2), (\neg z_1, \neg z_2) \notin \mathcal{C}_\Phi \ or \ (z_1, \neg z_2), (\neg z_1, z_2) \notin \mathcal{C}_\Phi \}.$$

The proof is obtained by showing that an arbitrary model for a true PCNF can be transformed to exhibit all the required independencies. We begin by defining an operation $\mathrm{ref}_u(P)$, which reforms a model path P based on the assignments of its complementary path with respect to a given universal variable u. We then prove that the resulting path does not falsify any clauses. For the remainder of this section, we extend the notion of exhibition of independence from pairs to sets of pairs; if $S = \{(z_1, z'_1), \ldots, (z_n, z'_n)\}$ we take $M \prec S$ to mean $M \prec (z_i, z'_i)$ for $i \in [n]$.

Definition 14 (Reformed path). *Let M be a model for a PCNF Φ over variables V, let $P \in M$, let $u \in V_\forall$ and put $l_u = P[u]$. The reformed path $\mathrm{ref}_u(P)$ of P with respect to u is given by*

$$\mathrm{ref}_u(P)[z] = \begin{cases} P_u[z] & \text{if } z \in V_\exists, P_u[z] = l_z, \text{ and } (\neg l_u, \neg l_z) \notin \mathcal{C}_\Phi\,, \\ P[z] & \text{otherwise}\,. \end{cases}$$

Lemma 15. *Let M be a model for a PCNF $\Phi = \mathcal{Q} \, . \, \phi$ over variables V, let $P \in M$ and let $u \in V_\forall$. Then $\mathrm{ref}_u(P)(C) = \top$ for all $C \in \phi$.*

Proof. Towards a contradiction, suppose that $\mathrm{ref}_u(P)(C) = \bot$ for some $C \in \phi$, and assume without loss of generality that $\mathrm{ref}_u(P)[u] = \neg u$ and $P_u[u] = u$. Since M is a model for Φ and $P \in M$, there is an existential literal $l \in C$ for which $P[\mathrm{var}(l)] = l$ but $\mathrm{ref}_u(P)[\mathrm{var}(l)] = \neg l$, so by Definition 14 we have $P_u[\mathrm{var}(l)] = \neg l$

and $(u, l) \notin \mathcal{C}_\Phi$. The latter implies that $(u, \neg l') \notin \mathcal{C}_\Phi$ for all existential literals $l' \in C$ such that $l' \neq l$. Hence, by Definition 14, if $P_u[\mathrm{var}(l')] = l'$, then $\mathrm{ref}_u[\mathrm{var}(l')] = l'$; but $\mathrm{ref}_u[\mathrm{var}(l')] = \neg l'$, so we must have $P_u[\mathrm{var}(l')] = \neg l'$. It follows that P_u falsifies all existential literals in C. Since $\mathrm{ref}_u(P)$ and P_u agree on all universal variables except u, and literal $u = P[u] \notin C$ (because $(u, l) \notin \mathcal{C}_\Phi$), P_u also falsifies all universal literals in C. Therefore $P_u(C) = \bot$, contradicting the premise that M is a model for Φ.   □

We proceed to define $\mathrm{ref}_u(M)$, the extension of the reformation operation from paths to models, in which all pairs of complementary paths P and P_u in some model M are reformed. The resulting model enjoys the useful properties stated in the subsequent lemma.

Definition 16 (Reformed model). *Let M be a model for a PCNF Φ over variables V, let $u \in V_\forall$, let $G = \{P \in M \mid P[u] = \neg u\}$ and let $M' = (M \setminus G) \cup \hat{G}$, where $\hat{G} = \{\mathrm{ref}_u(P) \mid P \in G\}$. Then the* reformed model *of M with respect to u is $\mathrm{ref}_u(M) = (M' \setminus G') \cup \hat{G}'$, where $G' = \{P \in M' \mid P[u] = u\}$ and $\hat{G}' = \{\mathrm{ref}_u(P) \mid P \in G'\}$.*

Lemma 17. *Let M be a model for a PCNF Φ over variables V, and let $u \in V$. Then*

(a) $\mathrm{ref}_u(M)$ *is a model for Φ,*
(b) $\mathrm{ref}_u(M) \prec \bar{\mathcal{D}}_\Phi^{\mathrm{rrs}}(u)$, *and*
(c) *if $M \prec (u', x)$, with $u' \in V_\forall$ and $(u', u) \in R_\Phi$, then $\mathrm{ref}_u(M) \prec (u, x)$.*

Proof. Let M' be defined as in Definition 16, and use the alias $M'' = \mathrm{ref}_u(M)$. Let $P \in M$ such that $P[u] = \neg u$ and let U be the set of universal literals in P. We denote by $P' \in M'$ and $P'' \in M''$ the unique paths with $U \subseteq P'$ and $U \subseteq P''$. Observe that, by Definition 16, $P'' = P' = \mathrm{ref}_u(P)$, $P''_u = \mathrm{ref}_u(P'_u)$ and $P'_u = P_u$.

(a) We prove that M' is a model for Φ. By Lemma 15, every path in M' satisfies ϕ. To show that M' is well-formed for Φ, let $v \in V_\forall$, let $U' = L_\phi(v) \cap V_\forall$ and let $S' \in M'$ such that $S'[v'] = P'[v']$ for all $v' \in U'$. Let $x \in V_\exists$ such that $x \in L_\phi(v)$, and let $S \in M$ such that $S' = \mathrm{ref}_u(S)$. Since M is well-formed and $S_u[v'] = P_u[v']$ for all $v' \in U'$, we must have $S_u[x] = P_u[x]$ for all $x \in V_\exists \cap L_\phi(v)$. Then $S'[x] = P'[x]$ by Definition 14. By construction, every universal assignment defines a unique path in M', so M' is well-formed for Φ. A similar argument shows that M'' is a model for Φ.

(b) Let $x \in V_\exists$ such that $(u, x) \notin \mathcal{D}_\Phi^{\mathrm{rrs}}$. It is sufficient to show that $P''[x] = P''_u[x]$ follows from Definition 14; to do this we consider two cases. (1) Suppose that $P[x] = P_u[x] = l_x$. Then $P'[x] = P'_u[x] = l_x$ and $P''[x] = P''_u[x] = l_x$. (2) Suppose instead that $P[x] = l_x$ and $P_u[x] = \neg l_x$. Since $(u, x) \notin \mathcal{D}_\Phi^{\mathrm{rrs}}$, we must have either $(u, l_x) \notin \mathcal{C}_\Phi$ or $(\neg u, \neg l_x) \notin \mathcal{C}_\Phi$. If $(u, l_x) \notin \mathcal{C}_\Phi$, we have both $P'[x] = P'_u[x] = \neg l_x$ and $P''[x] = P''_u[x] = \neg l_x$. On the other hand, if $(u, l_x) \in \mathcal{C}_\Phi$, $P'[x] = l_x$ and $P'_u[x] = \neg l_x$, whereupon $(\neg u, \neg l_x) \notin \mathcal{C}_\Phi$ yields $P''[x] = P''_u[x] = l_x$.

(c) Suppose that $M \prec (u', x)$, with $u' \in V_\forall$ and $(u', u) \in R_\Phi$, and put $Q = P_{u'}$. Similarly as for the path P above, let $U_{u'}$ be the set of universal literals in Q, and denote by $Q' \in M'$ and $Q'' \in M''$ the unique paths such that $U_{u'} \subseteq Q'$ and $U_{u'} \subseteq Q''$. Observe that, again by Definition 16, $Q'' = Q' = \mathrm{ref}_u(Q)$, $Q''_u = \mathrm{ref}_u(Q'_u)$ and $Q'_u = Q_u$. Since we assume $P[u] = \neg u$, to deduce $M'' \prec (u', x)$ we must show that $P''[x] = Q''_u[x]$ and that $P''_u[x] = Q''_u[x]$. The observation that $P[x] = Q[x]$ and $P_u[x] = Q_u[x]$, in combination with Definition 16, leads easily to the result. Firstly, this guarantees that $\mathrm{ref}_u(P)[x] = \mathrm{ref}_u(Q)[x]$, that is $P'[x] = Q'[x]$, therefore $P''[x] = Q''[x]$. Secondly, using $P'[x] = Q'[x]$, it also guarantees that $\mathrm{ref}_u(P'_u)[x] = \mathrm{ref}_u(Q'_u)[x]$, that is $P''_u[x] = Q''_u[x]$. $\square$

The main result of this section follows quickly.

Theorem 18. $\mathcal{D}^{\mathrm{rrs}}$ *is fully exhibited.*

Proof. Let M_0 be a model for a PCNF Φ over variables V, let $V_\forall = \{u_1, \ldots, u_n\}$ with $u_i <_\Phi u_{i+1}$ for $i \in [n-1]$, and let $M_{i+1} = \mathrm{ref}_{u_i}(M_i)$ for $i \in [n-1]$. We claim that M_n is a model for Φ such that $M_n \prec \mathcal{D}^{\mathrm{rrs}}_\Phi(u)$ for all $u \in V_\forall$.

By induction on $i \in [n]$, we prove that M_i is a model for Φ such that $M_i \prec \bigcup_{j=1}^{i} \bar{\mathcal{D}}^{\mathrm{rrs}}_\Phi(u_j)$, and hence at step $i = n$ we prove the claim and the theorem. For the base case $i = 1$, observe that M_1 is model for Φ by Lemma 17(a), and that $M_1 \prec \bar{\mathcal{D}}^{\mathrm{rrs}}_\Phi(u_1)$ by Lemma 17(b). For the inductive step, let $i \in [n-1]$ and suppose that M_i is a model for Φ and that $M_i \prec \bigcup_{j=1}^{i} \bar{\mathcal{D}}^{\mathrm{rrs}}_\Phi(u_j)$. Then $M_{i+1} = \mathrm{ref}_{u_i}(M_i)$ is a model for Φ by Lemma 17(a), $M_{i+1} \prec \bigcup_{j=1}^{i} \bar{\mathcal{D}}^{\mathrm{rrs}}_\Phi(u_j)$ by Lemma 17(c), and $M_{i+1} \prec \bar{\mathcal{D}}^{\mathrm{rrs}}_\Phi(u_{i+1})$ by Lemma 17(b). Therefore $M_{i+1} \prec \bigcup_{j=1}^{i+1} \bar{\mathcal{D}}^{\mathrm{rrs}}_\Phi(u_j)$. $\square$

Our concluding result now follows immediately from Theorems 8, 9 and 10.

Corollary 19. $\mathsf{QU}(\mathcal{D}^{\mathrm{rrs}})$-Res, $\mathsf{LD\text{-}Q}(\mathcal{D}^{\mathrm{rrs}})$-Res *and* $\mathsf{LQU}(\mathcal{D}^{\mathrm{rrs}})$-Res *are sound.*

6   Conclusions and Open Problems

As we have shown, the parametrisation by dependency schemes can be extended to all four CDCL QBF calculi, and the property of full exhibition – which is possessed by the reflexive resolution path dependency scheme – is sufficient for soundness in each case. Showing by counterexample that full-exhibition is not a necessary condition, our work leads naturally to the open problem of finding a characterisation for soundness in this setting. Another interesting question concerns proof complexity. The practical motivation to incorporate schemes into QBF solvers suggests that the use of suitable dependencies will shorten proofs. While it is not difficult to construct artificial schemes that yield a speed-up on specific formulas, the real question would be to understand the proof complexity impact of natural schemes like $\mathcal{D}^{\mathrm{rrs}}$.

References

1. Balabanov, V., Jiang, J.R.: Unified QBF certification and its applications. Formal Methods Syst. Des. **41**(1), 45–65 (2012)
2. Balabanov, V., Jiang, J.R., Janota, M., Widl, M.: Efficient extraction of QBF (counter)models from long-distance resolution proofs. In: Conference on Artificial Intelligence (AAAI), pp. 3694–3701 (2015)
3. Balabanov, V., Widl, M., Jiang, J.-H.R.: QBF resolution systems and their proof complexities. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 154–169. Springer, Heidelberg (2014)
4. Beyersdorff, O., Bonacina, I., Chew, L.: Lower bounds: from circuits to QBF proof systems. In: Proceedings of the ACM Conference on Innovations in Theoretical Computer Science (ITCS 2016), pp. 249–260. ACM (2016)
5. Beyersdorff, O., Chew, L., Janota, M.: On unification of QBF resolution-based calculi. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 81–93. Springer, Heidelberg (2014)
6. Beyersdorff, O., Chew, L., Janota, M.: Proof complexity of resolution-based QBF calculi. In: International Symposium on Theoretical Aspects of Computer Science (STACS). Leibniz International Proceedings in Informatics (LIPIcs), vol. 30, pp. 76–89 (2015)
7. Beyersdorff, O., Chew, L., Mahajan, M., Shukla, A.: Feasible interpolation for QBF resolution calculi. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9134, pp. 180–192. Springer, Heidelberg (2015)
8. Beyersdorff, O., Chew, L., Mahajan, M., Shukla, A.: Are short proofs narrow? QBF resolution is not simple. In: Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS 2016) (2016)
9. Lonsing, F., Biere, A.: Integrating dependency schemes in search-based QBF solvers. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 158–171. Springer, Heidelberg (2010)
10. Bubeck, U.: Model-based Transformations for Quantified Boolean Formulas. Ph.D. thesis (2010)
11. Egly, U.: On sequent systems and resolution for QBFs. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 100–113. Springer, Heidelberg (2012)
12. Egly, U., Lonsing, F., Widl, M.: Long-distance resolution: proof generation and strategy extraction in search-based QBF solving. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 291–308. Springer, Heidelberg (2013)
13. Janota, M., Marques-Silva, J.: Expansion-based QBF solving versus Q-resolution. Theorical Comput. Sci. **577**, 25–42 (2015)
14. Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified boolean formulas. Inf. Comput. **117**(1), 12–18 (1995)
15. Lonsing, F.: Dependency Schemes and Search-Based QBF Solving: Theory and Practice. Ph.D. thesis, Johannes Kepler University (2012)
16. Lonsing, F., Egly, U.: Incrementally computing minimal unsatisfiable cores of QBFs via a clause group solver API. In: Heule, M., et al. (eds.) SAT 2015. LNCS, vol. 9340, pp. 191–198. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24318-4_14
17. Samer, M.: Variable dependencies of quantified CSPs. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 512–527. Springer, Heidelberg (2008)

18. Samer, M., Szeider, S.: Backdoor sets of quantified boolean formulas. J. Autom. Reasoning **42**(1), 77–97 (2009)
19. Samulowitz, H., Bacchus, F.: Using SAT in QBF. In: International Conference on Principles and Practice of Constraint Programming (CP), pp. 578–592 (2005)
20. Slivovsky, F.: Structure in #SAT and QBF. Ph.D. thesis, Vienna University of Technology (2015)
21. Slivovsky, F., Szeider, S.: Soundness of Q-resolution with dependency schemes. TCS **612**, 83–101 (2016)
22. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time: preliminary report. In: Annual Symposium on Theory of Computing, pp. 1–9. ACM (1973)
23. Van Gelder, A.: Variable independence and resolution paths for quantified boolean formulas. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 789–803. Springer, Heidelberg (2011)
24. Van Gelder, A.: Contributions to the theory of practical quantified boolean formula solving. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 647–663. Springer, Heidelberg (2012)
25. Zhang, L., Malik, S.: Conflict driven learning in a quantified boolean satisfiability solver. In: International Conference on Computer-aided Design (ICCAD), pp. 442–449 (2002)

The Multirate Resource Constraint

Alessio Bonfietti[1]([✉]), Alessandro Zanarini[2], Michele Lombardi[1],
and Michela Milano[1]

[1] DISI, University of Bologna, Bologna, Italy
{alessio.bonfietti,michela.milano,michele.lombardi2}@unibo.it
[2] ABB Corporate Research Center, Baden, Switzerland
alessandro.zanarini@ch.abb.com

Abstract. Many real world cyclic scheduling problems involve applications that need to be repeated with different periodicity. For example, multirate control systems present multiple control loops that are organized hierarchically: the higher-level loop responds to the slower system dynamics and typically its period can be a few orders of magnitude longer than the lowest level. Cyclic scheduling problems can be cast into classical RCPSP instances via a technique called unfolding [4,6], which causes graph expansion. In the case of multirate applications, this expansion can be significantly large. In this context, finding a high-quality allocation and schedule could be very challenging. In this paper, we propose a new *Multirate Resource Constraint*, modeling unary resources, that avoids graph expansion by exploiting the multirate nature of the schedule in its filtering algorithm. In an experimentation on synthetic and real-world instances, we show that our method drastically outperforms approaches based on state-of-the-art unfolding and constraint based scheduling.

1 Introduction

The increasing number of functionalities delivered in advanced control solutions employed in different domains (e.g. process automation, automotive industry and so on) requires more and more hardware resources in order to compute optimal control strategies to feed to the controlled system. The number of functional blocks forming such advanced control solutions can be in the order of thousands or ten of thousands and they are organized in hierarchical feedback loops. The higher levels (or outer loop) provide supervisory control functions for high-level management and the setpoints for the lower levels; the middle levels feature direct process control of the plant or system actuators; the lower levels (or inner loop) typically present the fail-safe, protection and safety functions ensuring that the plant remains within specifications avoiding possibly irreversible responses. All the levels run periodically: the inner loops have the highest priority and very short periods (in order to respond to the fast dynamics of the system) and the outer loops have the lowest priority and much longer periods (in order to address the slower dynamics of the system). Such approach is typically referred to as *multirate control system*.

In the recent past, the embedded hardware running the control solutions offered a single CPU on which all the control loops were executed. Nowadays, control companies are introducing additional computational power under the form of multi-core CPUs (or system-on-a-chip presenting also FPGA or DSPs). The control engineer is therefore facing new challenges as he needs to master not only the control technologies and underlying mathematical algorithms, but also the deployment of the control solutions on multiple - possibly heterogeneous - computational resources. Optimizing the allocation and scheduling of the control functions on multi-core heterogeneous hardware becomes key for leveraging the power offered by such hardware architectures (see for example [7, 12, 16–18]).

From an optimization viewpoint, the challenge of the resource allocation and multirate periodic scheduling problem resides on the fact that the longest control loop period may easily be two or three orders of magnitude longer than the shortest period. Consequently, tackling the problem as a whole can be computationally very demanding.

The contribution of this paper is twofold: we present a constraint programming model for the multirate periodic scheduling problems and we introduce a new global constraint - the *multirate resource constraint* - that captures the specific sub-structure of multirate systems. The paper is organized as follows: Sect. 2 formalizes the problem; in Sect. 3 we present the model employed to solve the problem; in Sect. 4 we introduce the novel multirate cumulative constraint and the related filtering algorithm; Sect. 5 shows the experimental results and finally conclusions are drawn in Sect. 6.

2   Problem Description

The problem at hand consists of a resource allocation and multirate periodic scheduling on r *unary* resources, where $r \in \mathbb{R}$. For the rest of the paper, we will refer to the control feedback loops as applications.

A set of m periodic applications $\mathtt{A} : \{a_0, \dots, a_{m-1}\}$ is given. Each application a_i has a priority value, corresponding to the index i (where $i = 0$ stands for the highest priority and $i = m - 1$ for the lowest), and fixed period λ_i. Each application a_{i+1} has a period λ_{i+1} multiple of the period λ_i of the application a_i. Formally, $\lambda_{i+1} = \eta \cdot \lambda_i$ for some $\eta \in \mathbb{N}^+$. The application with the lowest priority has the longest period, which is called the *reference* period and is noted as λ_{max}. The application with the highest priority has the shortest period, which is called the *base* period. Since different applications can have periods of different lengths, two applications may execute a different number of times within the same time window. Each execution of an activity is called *repetition*. Considering λ_{max} as the time window, an application a_i will have $rep(a_i) = \frac{\lambda_{max}}{\lambda_i}$ repetitions.

Each application a_i is composed by a different set of activities $\mathtt{V}_i$: $\{\mathtt{x}_0^i, ..., \mathtt{x}_{n-1}^i\}$; $\mathtt{V}$ is the set of all activities $\mathtt{V} = \cup_{i=0,\dots,m-1}\mathtt{V}_i$. Note that each application has its own number of activities. In the most general case, the duration of an $\mathtt{x}_j^i$ can be allocation-dependent, i.e. it may vary depending on which resource r it is assigned to. In this work each activity $\mathtt{x}_j^i$ has a fixed duration

$d(\mathbf{x}^i_{jr})$ for each resource $r \in \mathtt{R}$. We assume that there are no transition times between two activities running on the same resource and we discard the communication overhead of two activities running on two different resources. Please note that the multirate resource constraint introduced in Sect. 4 can be easily extended to account both for allocation-dependent activity durations and transition times. Finally, activities within the same application can be subject to a set of precedence constraints where $\mathbf{x}^i_q \prec \mathbf{x}^i_k$ indicates that activity q must finish before activity k starts. We assume that there are no precedence constraints across different applications. Finally, the objective is to minimize the lexicographic composition of all the application makespans, i.e. to minimize first the makespan of a_0, followed by a_1 and so on.

3   Constraint Model and Search Strategy

Each activity $\mathbf{x}^i_j$ is modeled with a set of $|\mathtt{R}|$ (i.e. one for each resource $r \in \mathtt{R}$) conditional interval variables $\mathbf{s}^i_{jr} : [0, \lambda_i - d(\mathbf{x}^i_{jr})]$, where $\underline{\mathbf{s}}^i_{jr}$ and $\overline{\mathbf{s}}^i_{jr}$ are the lower and the upper bound, respectively. The variable $\mathbf{s}^i_{jr}$ models the j-th activity of the i-th application if executed on resource r. The conditional interval variables, introduced in [8,9], can easily model activities that can execute on a set of alternative resources with different execution times. Each conditional interval variable has an execution status $ex(\mathbf{s}^i_{jr})$, which is equal to 1 if the variable is executed, or equal to 0 if the variable is non-executed. The concept of execution status is strictly related to the allocation of an activity on a resource r. We can therefore model the allocation with the following constraint which states that each activity must be executed on a single resource:

$$\sum_{r \in \mathtt{R}} ex(\mathbf{s}^i_{jr}) = 1 \qquad \forall i \in \mathtt{A} \text{ and } \forall j \in \mathtt{V}_i \qquad (1)$$

In the paper, the notation $\mathbf{s}^i_{jr}$ will be used to refer both to the interval variable and to its associated start time. The set of the interval variables for the activities of the application a_i is $\mathtt{S}_i : \{\mathbf{s}^i_{0,r}, ..., \mathbf{s}^i_{n-1,r}\} \quad \forall r \in \mathtt{R}$, while $\mathtt{S}$ is the set of all the interval variables, i.e. $\mathtt{S} = \bigcup_i \mathtt{S}_i \quad \forall \, 0 \leq i < m$. An interval variable is *bound* if a decision has been taken, namely its start time value and execution status have been fixed; otherwise the variable is called *free* (or *unbound*).

Note that the applications are periodic, therefore we can focus only on the start time $\mathbf{s}$ of the activities belonging to the first repetition; all the execution times of the remaining repetitions can be derived by shifting the first one by ω times the period. Therefore the start time of first and the ω-th repetition are:

$$start(\mathbf{x}^i_j, 0) = \mathbf{s}^i_{jr} \quad \text{and} \quad start(\mathbf{x}^i_j, \omega) = \mathbf{s}^i_{jr} + \omega \cdot \lambda_i$$

The precedence constraints between activities are modeled through classical temporal constraints where the start time of the sink activity cannot be lower than the end time of the source activity. If $(\mathbf{x}^i_j, \mathbf{x}^i_{j'})$ is a precedence from $\mathbf{x}^i_j$ to $\mathbf{x}^i_{j'}$ we model the constraint as follows:

$$\mathbf{s}^i_{jr} + d(\mathbf{x}^i_{jr}) \leq \mathbf{s}^i_{j'r'} \qquad \forall r, r' \in \mathtt{R} \qquad (2)$$

The limited resource capacities are handled with the **Multirate Resource Constraint** (MRC). Its definition and propagation algorithm are described in Sect. 4. It is indeed possible to use traditional resource constraints (e.g. both unary [15] and cumulative [13]), but only at the price of modeling *each repetition of each activity* via a distinct variable. By exploiting the intrinsic periodicity of the problem and taking advantage of modular arithmetic, the MRC constraint allows us to model all the repetitions with a single variable.

The objective is to minimize the makespan of each application (considering the first repetition), where $MK_i = \max(s^i_{jr} + d(x^i_{jr}))$ is the makespan of the application a_i; namely it is the distance between the greatest end time and the initial value 0. Since one of the restrictions imposed by the problem is that the execution of an activity of a high priority application cannot be delayed for the execution of an activity of a lower priority application, we decided to decompose the problem into a sequence of optimization subproblems. Each subproblem is solved to optimality and consists of the minimization of the makespan of a single application. The subproblems are solved in order of application priority (i.e. from the smallest period to the largest one). Note that the model of the i-th subproblem has to include the variables of the previously solved subproblems (i.e. with $i' < i$). The activities of the applications of each solved subproblem are crucial for the resource constraint filtering and are modeled through initially *bound* start time variables. If a traditional resource constraint is used, each of those activities has to be replicated several times, depending on the multiplying factor between the periods. As the experiments highlight, this causes an explosion in the model size and, as a consequence, in the number of variables that a resource constraint must handle. Conversely, by using the Multirate Resource Constraint our approach requires only the first execution of each activity, drastically reducing the size of the model.

3.1 The Search

We have developed an ad-hoc search strategy, called *precedence* search.

The strategy aim is to avoid the choice of activities whose predecessors have not yet been scheduled. This is done by carrying out a preliminary search in the list of activities to be scheduled, and populating a list called *ready*, with only the activities that, if scheduled, do not violate the precedence constraints. In other words, the list is populated with activities whose predecessors have already been scheduled. The choice of the activity is then carried out by selecting the one that appears on top of the *ready* list (in FIFO order).

The algorithm then attempts to schedule the activity by choosing, between its interval variables (one for each resource), the one that can be executed with the lowest earliest start time, assigning it to its minimum value. On backtrack, the activity is postponed, i.e. marked as non selectable until its earliest start time is modified by propagation. This is analogous to what is done in the classical schedule-or-postpone strategy for the RCPSP [10] and can be done since we enforce resource restrictions by means of the Multirate Resource Constraint (which filters on the lower bounds).

4   The Multirate Resource Constraint

The Multirate Resource Constraint MRC has the following signature:

$$\texttt{MRC}([\texttt{s}_{\texttt{jr}}^{\texttt{i}}], [\texttt{d}(\texttt{x}_{\texttt{jr}}^{\texttt{i}})], [\lambda_{\texttt{i}}], [\texttt{V}_{\texttt{i}}])$$

where $[\mathbf{s}_{jr}^i]$ is a vector of (conditional) interval variables, $[d_{jr}^i]$ is the vector of corresponding durations, $[\lambda_i]$ and $[V_i]$ are the period and the set of activities for application i. In the rest of the paper, as the constraint models a single resource, we refer to the variables and the durations as $\mathbf{s}_j^i$ and $d(x_j^i)$, respectively.

The aim of the filtering algorithm is to update the Earliest Start Time (EST) of each interval variable by eliminating time points where the available resource capacity is not sufficient. Similarly to timetable filtering [1], ou algorithm prunes the EST so that it corresponds *the first time point where there is no conflict with the compulsory parts of other activities*. Note that this is weaker then Bound Consistency, which would be NP-hard to enforce for the MRC as it is the case for the classical cumulative constraint.

An activity x_j^i has a compulsory part if and only if there exists a time span where the activity is necessarily executing. This happens if the latest start time (i.e. LST) of a start time variable is smaller than its earliest end time (i.e. EET), where the EET is the earliest start time summed with the activity duration: $EST_j^i + d(x_j^i) > LST_j^i$.

In the first subsection we define some concepts and some operators used in the algorithm itself. Then we presents the main rules and the steps of the algorithm. We conclude the section with an example. In the rest of this section we suppose that we are filtering on the variable $\mathbf{s}_j^i$ modeling the activity x_j^i of the application a_i on the resource of the constraint; we call it the *selected* variable.

4.1   Definitions

Here we present some definitions used in the rest of the section.

Definition 1. *Let t be a time point, $\Delta(t)$ is the set of applications having a period ending at time t, formally:*

$$\Delta(t) = \{ \ a_i \ \mid \ t \bmod \lambda_i = 0\}$$

We can now define the modularization of a variable[1].

Definition 2. *The modularization $m_k(\mathbf{s}_j^i)$ w.r.t. period λ_k is defined on the bounds of the variable $\mathbf{s}_j^i$ as follows:*

$$\underline{m}_k(\mathbf{s}_j^i) = \underline{\mathbf{s}}_j^i \bmod \lambda_k$$
$$\overline{m}_k(\mathbf{s}_j^i) = \overline{\mathbf{s}}_j^i \bmod \lambda_k$$

---

[1] In the whole paper we assume that the durations and the periods are positive values.

Note that if the duration of an activity $\mathbf{x}_j^i$ is longer or equal than the period of another application a_k (i.e. $d(\mathbf{x}_j^i) \geq \lambda_k$), no activities of the application a_k can be scheduled since they would conflict with $\mathbf{x}_j^i$, and the constraint is infeasible.

Classical and modular resource constraints (such as [3] and [5], respectively) run their filtering algorithms on a time window coinciding with the horizon of the schedule, in our case the reference period λ_{max}. The key idea of our algorithm is to exploit the potential of the periodicity, focusing on time windows of increasing size and considering subsets of activities.

Definition 3. *Given a selected variable $\mathbf{x}_j^i$, we define Λ_k as the resource profile over the period λ_k (with $k \leq i$) accounting for the compulsory parts of the following activities:*

$$\mathbf{V}_k \qquad\qquad\qquad if\ k < i \qquad\qquad (3)$$

$$\bigcup_{k'=i,\dots,m-1} \mathbf{V}_{k'} \qquad\qquad if\ k = i \qquad\qquad (4)$$

The algorithm processes the profile Λ_k by increasing k (therefore with increasing period size) up to Λ_i, the level of the application of the *selected* activity $\mathbf{x}_j^i$ ($k = 0, .., i$). Each resource profile Λ_k with $k < i$ accounts only for the compulsory parts of activities $\mathbf{V}_k$.

When Λ_i is considered, activities belonging to applications with longer period ($k > i$) may not appear within the first repetition of $\mathbf{x}_j^i$ and yet conflict with *subsequent* repetitions of $\mathbf{x}_j^i$. Therefore, they should be taken into account in a modularized manner (see Definition 2) with the respect to the period λ_i of the application a_i. In other words, while the algorithm is performing filtering on $\mathbf{s}_j^i$, it has to avoid that a start time t' is considered feasible for an activity x_j^i if there exists an activity $x_{j'}^{k'}$, belonging to an application having a longer period (i.e. $k' > i$), whose execution is in conflict with the ω-th repetition of the activity x_j^i. Formally if at least one of the following conditions is verified there exists a conflict:

$$\mathbf{s}_j^i + \omega \cdot \lambda_i \leq \mathbf{s}_{j'}^{k'} < \mathbf{s}_j^i + d(\mathbf{x}_j^i) + \omega \cdot \lambda_i, \quad \forall \omega \in \mathbb{N}^+ \qquad (5)$$

$$\mathbf{s}_j^i + \omega \cdot \lambda_i < \mathbf{s}_{j'}^{k'} + d(\mathbf{x}_{j'}^{k'}) \leq \mathbf{s}_j^i + d(\mathbf{x}_j^i) + \omega \cdot \lambda_i, \quad \forall \omega \in \mathbb{N}^+ \qquad (6)$$

$$\mathbf{s}_j^i + \omega \cdot \lambda_i \geq \mathbf{s}_{j'}^{k'} \ \mathrm{AND} \ \mathbf{s}_{j'}^{k'} + d(\mathbf{x}_{j'}^{k'}) \geq \mathbf{s}_j^i + d(\mathbf{x}_j^i) + \omega \cdot \lambda_i, \quad \forall \omega \in \mathbb{N}^+ \qquad (7)$$

Consider as example Fig. 1. The schedule depicts a state of the profile of a resource during search. The problem is composed by three applications with
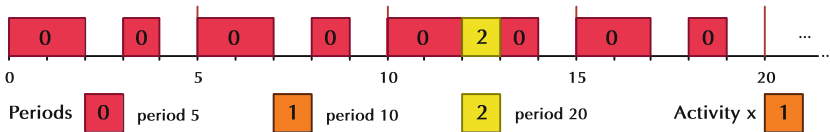
Fig. 1. Example with a partial solution. (Color figure online)

different periods denoted by colors and numbers. Suppose that we are filtering on a variable $\mathbf{s}_j^1$ of the Orange(1) application (having period $\lambda_1 = 10$) and that the variable has execution time 1. The first feasible start time could erroneously considered to be 2, in fact there is a time window of length 1 between the first executing activity and the next one. Unfortunately an activity $\mathbf{s}_{j'}^2$ of the Yellow(2) application, which has longer period (i.e. $\lambda_2 = 20$), is scheduled at time 12 (i.e. $\mathbf{s}_{j'}^2 = 12$). This conflict is identified by both conditions (5) and (7), where $\mathbf{s}_{j'}^{k'}$ is $\mathbf{s}_{j'}^2$, the activity of the Yellow(2) application, and $\mathbf{s}_j^i$ is the selected activity $\mathbf{s}_j^1$ (and $\omega = 1$). In fact, if the activity $\mathbf{s}_j^1$ would be scheduled at 2 (i.e. $\mathbf{s}_j^1 = 2$) during its second repetition it would conflict with $\mathbf{s}_{j'}^2$, since $\mathbf{s}_j^1 + \lambda_1 = 2 + 10 = 12$ (recall that $\lambda_1 = 10$).

We can now define a feasibility function used in the filtering algorithm.

Definition 4. *Let* $feas(\mathbf{s}_j^i, k, t)$ *be a function that checks if an activity* x_j^i *can be scheduled at time* t.

$$feas(\mathbf{s}_j^i, k, t) = <[true\,|\,false],\ \pi > where\,\pi \geq t$$

The function returns a pair composed by a boolean value and a time instant. The boolean is true iff t *is feasible when considering resource profiles* $\Lambda_0, \ldots, \Lambda_k$. *If the boolean is false, the time instant* π *corresponds to the maximum end time between all the activities conflicting with* x_j^i *(i.e. the first feasible start time).*

The function $feas(\mathbf{s}_j^i, k, t)$ checks the feasibility of t in each period $\lambda_{k'}$ with $k' \leq k$; this can be done efficiently by computing $t \mod \lambda_k$ and checking if the modularized time instant overlaps with a compulsory part in $\Lambda_{k'}$. As explained above, if $k = i$, the function considers Λ_i by modularizing the compulsory parts of activities belonging to applications with longer period ($k' > i$).

In the worst case this function has to check $\bar{n} - 1$ activities ($\bar{n}$ is the total number of the activities) hence its asymptotic computational complexity is $\mathcal{O}(\bar{n})$.

Let $O = \{\mathbf{o}_0, .., \mathbf{o}_{m-1}\}$ be a set storing a value for each application a_i; these values are referred to as *offsets*.

Definition 5. *The offset* $\mathbf{o}_i = [0..\lambda_i]$ *is a value representing the minimum feasible start time considering the resource conflicts arising from resource profiles* $\Lambda_0, \ldots, \Lambda_i$.

Intuitively it represents the minimum feasible start time within each period: $\mathbf{o}_i = 2$ means that in each repetition ω of application a_i, no activity can be scheduled before the instant $2 + \omega \cdot \lambda_i$. Note also that the offsets are non-decreasing ($\mathbf{o}_{i+1} \geq \mathbf{o}_i$) as the set of activities considered in level i is a subset of the activities in level $i + 1$ ($\bigcup_{0 \leq k \leq i} \mathbf{a}_k \subseteq \bigcup_{0 \leq k \leq i+1} \mathbf{a}_k$).

4.2   Algorithm Rules

Let σ be a time value (also referred to as sweeping-line in the literature [11]) used to compute the minimum feasible start time for the selected activity (i.e. x_j^i).

In the following we illustrate the rules used in the filtering algorithm in order to update the sweeping line σ and the offsets O. The algorithm leverages three rules declaring how to update the offsets and σ, how to recognize an infeasibility, and how to exploit the periodicity for jumping over large time windows with proven resource conflicts.

Rule 1. *Suppose the algorithm is considering level k, if the feasibility function returns a failure then the offset o_k and the sweeping line σ are updated to the time value returned by the feasibility function. Formally:*

$$feas(\mathbf{s}_j^i, k, t) = < false, \ \pi > \quad \Rightarrow \quad \sigma = o_k = \pi \tag{8}$$

Intuitively this rule states that if the selected activity cannot be scheduled at time t, then time π should be considered next as it coincides with the latest end time of the conflicting activities. Please note that Rule 1 is superseded by Rule 3 in the specific case the latter applies.

Rule 2. *If an offset is greater or equal to its relative period, the algorithm fails.*

$$o_k \geq \lambda_k \quad \Rightarrow \quad fail, \quad \forall k \mid a_k \in \mathbf{A} \tag{9}$$

If for the current activity there is no feasible space in level Λ_k, and this is due to conflicts with activities of applications having index $k' \leq k$, there will be no space at all, since the conflicts will be replicated periodically. Before defining the third rule, we explain how it intuitively works with an example. Consider Fig. 2 which is also used as example in Sect. 4.4. Suppose that we are filtering on an activity $\mathbf{s}_j^2$ with duration 2 of the Yellow(2) application (we recall that each application is depicted with a color and a number and has a different period). Suppose now that the algorithm has already found that before $t = 7$ there is no feasible space, and that this is due to conflicts with activities of the Red(0) and the Orange(1) applications (which have shorter periods w.r.t. the Yellow(2) one). The already scheduled Yellow(2) activity at time $t = 7$ prevents the selected activity x_j^2 to be scheduled. So the algorithm should, in principle, proceed, step by step, up to instant $t = 17$ in order to find a feasible time window. The algorithm can exploit the application periodicity and, as the first 7 instants of time of the period of the Orange(1) application contain conflicts, whenever the search crosses the end of the Orange(1) period (i.e. $\lambda_1 = 10$), it can jump directly to instant $10 + 7 = 17$. Formally:

Rule 3. *If an activity x_j^i cannot be scheduled within a period ending at t the sweeping line σ can be shifted forward to the maximum between:*
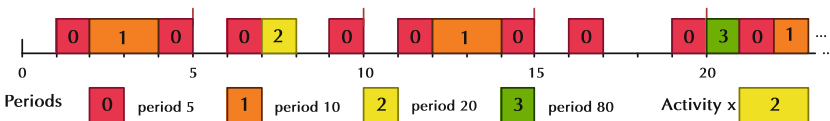
Fig. 2. Example with a partial solution. (Color figure online)

- $t + \mathsf{o}_{max}$, where o_{max} is the greatest offset of all the applications in $\Delta(t)$
- the time π computed by the feasibility function.

$$if\ feas(\mathsf{s}^i_j, \hat{k}, \sigma) = <false,\ \pi> \quad and \quad \pi \geq t \quad \forall\ \hat{k} \leq i \tag{10}$$

$$\Rightarrow \quad \sigma = \max(\pi,\ t + \mathsf{o}_{max}) \tag{11}$$

$$where \quad \mathsf{o}_{max} = \max(\{\mathsf{o}_{k'} \mid k' \in \Delta(t)\})$$

As a consequence the offset is updated: $\mathsf{o}_i = \sigma$.

Proof. From Definition 5 we recall that the offset defines, for each application a_i, the minimum feasible time instant. Its value is updated anytime the feasibility function returns a failure (see Rule 2).

Let $t = \omega \cdot \lambda_k$ be a time instant coinciding with the end time of the period of some applications. $\Delta(t)$ is the set of these applications. If their period is ending at t, $t+1$ is the first time instant of a new repetition of each application of $\Delta(t)$. We also know by definition that the first o_i-*th* time instants of each repetition of the period of the application a_i are infeasible. Focusing only on the offsets, we can therefore affirm that the longest infeasible time window starting at time t corresponds to the biggest offset between the ones related to the applications of $\Delta(t)$. Since $\mathsf{o}_{i+1} \geq \mathsf{o}_i$ and $\lambda_{i+1} \geq \lambda_i$, the biggest offset corresponds to the application of $\Delta(t)$ having the longest period.

From Definition 4, the feasibility function returns a time value π corresponding to the maximum end time between all the activities conflicting with the selected activity. We can therefore declare that the longest infeasible time window is the maximum between the value π computed by the function feas() and the biggest offset of the applications of $\Delta(t)$.

4.3   Algorithm Steps

The filtering algorithm executes for each free start time variable. These variables are stored in a queue ordered by application (increasing order of the indices). The pseudo-code of the algorithm is in Algorithm 1.

Initially we set (lines 2-3):

$$\sigma = \underline{\mathsf{s}}^i_j$$

$$and \quad \mathsf{o}_i = 0 \quad \forall i \mid a_i \in \mathsf{A}$$

As stated before, the algorithm proceeds focusing on one level at time with increasing size order (line 4). The starting value for the offset is the σ value (line 5), which represents the minimum time where the infeasibility has not been proved. In other words, the selected activity cannot be scheduled before σ. At line 6 the algorithm starts searching for a feasibility window. For each time t, starting from the initial value σ and up to the end of the current level k (i.e. λ_k), the algorithm checks the feasibility.

Algorithm 1. Consistency - Start Time Filtering Algorithm

    Data: *Let $\underline{s}_j^i$ be a free start time variable of the application a_i*

1 **begin**
       // Initialization
2      $\sigma \leftarrow \underline{s}_j^i$;
3      $o_i \leftarrow 0 \quad \forall i \mid a_i \in A$;
4   **for** $k = 0; k \leq i; k = k + 1$ **do**
         // For each level with no greater period
5          $o_k \leftarrow \sigma$;
6   **for** $t = \sigma; t < \lambda_k; t = t + 1$ **do**
7   Let $< \beta, \phi >$ be a pair $<$[true|false], time$>$ storing the result of feas();
8             $< \beta, \pi > \leftarrow$ feas(s_j^i, k, t);
9   **if** β *is* true **then**
               // Setting the new lowerbound
10               $s_j^i \geq \sigma$;
11   break;
12   **else**
               // t is not feasible
13   **if** $\pi \geq \lambda_k$ **then**
14   *fail()*;
15   Let $< \delta, k', t' >$ be a triple storing the result of *findPeriod()*;
16               $< \delta, k', t' > \leftarrow$ *findPeriod*(t, π);
17   **if** δ *is* true **then**
18                  $t \leftarrow \max(\pi, t' + o_{k'})$;
19   **else**
20                  $t \leftarrow \pi$;
21               $o_k \leftarrow t$;
22               $\sigma \leftarrow t$;

If t is feasible, the lower bound of the variable is updated and the algorithm proceeds to another level $k+1$ or it ends if $k = i$ (lines 9-11). On the other side, if t is not feasible, the algorithm applies the three rules.

At the beginning the Rule 2 is evaluated (lines 13-14). If the minimum feasible time π is not lower than the current period λ_k it means that there is no space for the activity and the constraint fails.

Then we have to check if the computed time π has crossed the end of some periods (line 16). The function findPeriod(t, t') returns a triple $<$[true|false], k, time$>$, where the first element states if between t and t' (with $t \leq t'$) there exists the end of at least a period, the second element is the index of the application with the longest period ending in the analyzed time window, and the third element is the period end time. Exploiting the modular algebra this is asymptotically done in $\mathcal{O}(m)$, where $m = |A|$. If the function returns a success the algorithm applies Rule 3 (lines 17-18) otherwise it executes the Rule 1 (line 19-20). At the end the offset and the σ values are updated.

4.4   Example

In this section we apply the filtering algorithm to the example depicted in Fig. 3. The schedule (A) shows a partial solution. We have 4 Applications: Red(0),

Orange(1), Yellow(2) and Green(3) with periods 5,10,20 and 80 respectively. We are filtering on $\mathbf{s}_0^2 : [0..18]$: a yellow(2) activity with duration 2. The algorithm initially sets $\sigma = 0$ and $\mathbf{o} = \{0, 0, 0, 0\}$.
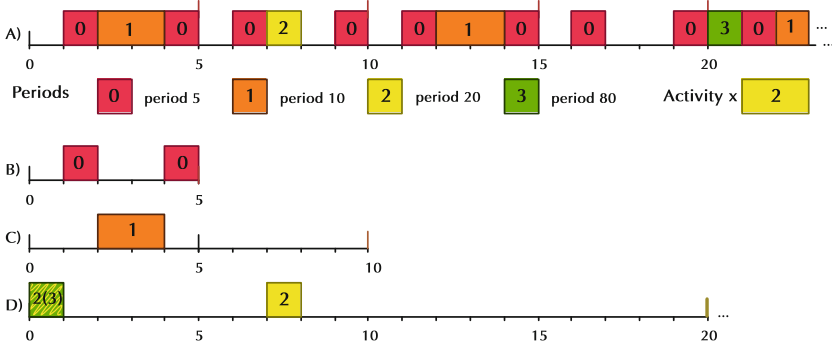
Fig. 3. Example with a partial solution (A) and the representation of three levels (0),(1),(2) as (B), (C), and (D), respectively. (Color figure online)

The method focuses on the resource profiles $\Lambda_0, \Lambda_1,$ and Λ_2 depicted in Fig. 3 (B), (C), and (D) respectively. Note that in the schedule (A), at time 20, there's an activity of the application Green(3) which has a period longer than the Yellow(2) one. This activity is modularized and folded back at time 0 in the profile (D) (20 mod λ_2). The algorithm starts by processing Λ_0 and checking whether σ is feasible for the activity x_j^i:

$$\text{feas}(\mathbf{s}_j^i, 0, 0) =< \text{false},\ 2 >$$

The result means that time 0 is infeasible, and that there exists at least a conflicting activity executing up to time 2 (the first Red(0) activity executing $1 \to 2$). The system therefore updates σ and the relative offset value (following Rule 1):

$$\sigma = 2 \qquad \mathbf{o} = \{2, 0, 0, 0\}$$

The algorithm reiterates on the feasibility check with the new σ:

$$\text{feas}(\mathbf{s}_j^i, 0, 2) =< \text{true},\ 2 >$$

and it finds that time 2 is feasible w.r.t. the resource profile Λ_0. The algorithm therefore proceeds with the resource profile Λ_1 starting at time $\sigma = 2$.

$$\text{feas}(\mathbf{s}_j^i, 1, 2) =< \text{false},\ 4 >$$

As an Orange(1) activity scheduled at 2 is conflicting, the function returns $false$. The method then applies Rule (1) and updates σ and the offset value:

$$\sigma = 4 \qquad \mathbf{o} = \{2, 4, 0, 0\}$$

A new feasibility check iteration returns a new conflict:

$$\text{feas}(\mathbf{s}_j^i, 1, 4) =< \text{false},\ 5 >$$

by checking in decreasing order the profiles Λ_1 and Λ_0, the activity Red(0) scheduled at 4 is found as conflicting. As the new $\pi = 5$ is equal to λ_0, Rule (3) is applied: $\sigma = \max(\lambda_0 + o_0, 5) = \max(5 + 2, 5) = 7$. In fact, the second repetition of the application Red(0) will cause infeasibility up to time 7. The relative offset value and σ are then updated accordingly:

$$\sigma = 7 \qquad \mathsf{o} = \{2, 7, 0, 0\}$$

A further feasibility check returns a positive outcome for time 7 w.r.t. resource profile Λ_1 and Λ_0:

$$\text{feas}(\mathsf{s}_j^i, 1, 7) = < \text{true, } 7 >$$

therefore the algorithm starts processing Λ_2 after updating the sweeping-line $\sigma = 7$. The feasibility function finds an activity of the Yellow(2) application executing in $7 \rightarrow 8$ as conflicting:

$$\text{feas}(\mathsf{s}_j^i, 2, 7) = < \text{false, } 8 >$$

consequently the filtering algorithm updates σ and the offset value:

$$\sigma = 8 \qquad \mathsf{o} = \{2, 7, 8, 0\}$$

Time 8 is again infeasible:

$$\text{feas}(\mathsf{s}_j^i, 2, 8) = < \text{false, } 10 >$$

due to the second repetition of a Red(0) activity. The time instant 10 coincides with the end time of two periods, namely the second repetition of the Red(0) application (i.e. $2 \cdot \lambda_0$) and the first repetition of the Orange(1) one (i.e. $1 \cdot \lambda_1$). Therefore, Rule (3) is applied again. The longest between the two periods is the Orange(1) one, hence its offset is used to updated the sweeping line:

$$\sigma = \max(1 \cdot \lambda_1 + o_1, 10) = \max(10 + 7, 10) = 17 \qquad o_2 = 17$$

Note that the algorithm was able to forward the sweeping-line by a time window of length 9 which is proved to contain conflicting parts and no space for scheduling. The feasibility function at time 17 returns a success: the algorithm stops and the domain of s_j^i is modified in [17..18].
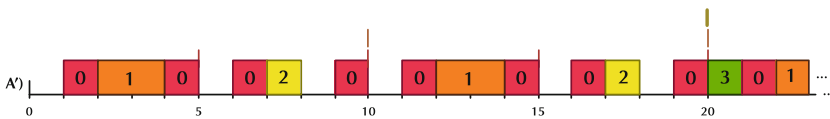
Fig. 4. Partial Solution representing an infeasible situation. (Color figure online)

Suppose now that an activity (with duration 1) of the application Yellow(2) is scheduled at 17 (see the schedule in Fig. 4). In this case the feasibility function at time 17 would have failed:

$$\text{feas}(\mathsf{s}_j^i, 2, 17) = < \text{false, } 18 >$$

because of the newly placed Yellow(2) activity. The function would have failed also at time 18 (due to the fourth repetition of a Red(0) activity) and the sweeping-line and offsets would have been updated as follows:

$$\sigma = 20 \qquad \mathsf{o} = \{2, 9, 20, 0\}$$

and, since $\mathsf{o}_2 = 20 \geq \lambda_2 = 20$, the filtering algorithm would have failed (following Rule (2)).

5   Experimental Results

The aim of the experimental section is to show that the proposed algorithm is efficient and scalable and outperforms both state of the art approaches and industrial solutions. The solver and the constraint have been implemented using the or-tools framework, supported and developed by Google [14]. The experiments can be structured into two parts: the former considers various sets of synthetic but realistic instances, the latter considers two industrial instances. The synthetic instances were built by means of an internally developed generator[2], designed to produce instances with realistic structure and parameters. Our generation algorithm builds application graphs which are connected, consistent, and cycle-free. A user specifies in a configuration file the instance parameters. In this work we have generated instances with two different graph structures: (1) graphs with a *sequential* structure, with few long chains, and (2) graphs with a more *parallel* form.

The Multirate Resource Constraint (referred to as M in the tables) is compared with two different resource constraints: the former (labelled T&E) is a combination of the Time Table and the Edge Finder resource constraints, while the latter (labelled DJ) is the Disjunctive resource constraint (see [2] for an overview of these algorithms). All the algorithms achieve the same filtering but substantially different performance on unary resources. For all the approaches we used the same ad-hoc search strategy, described in Sect. 3.1.

All the experiments presented are for problems with two resources, since the real-world instances are defined for two CPU cores. We also performed tests with more resources (i.e. 4 and 8), but they did not add much information since the overall trend was (scaled but) similar.

Synthetic Benchmarks. The experiments on synthetic benchmarks have been organized: the aim of the first experimental evaluation is to show the performance of the Multirate Resource Constraint in instances where (almost) all the approaches find the optimal solution within the time limit[3] (300 s). The second

---

[2] The generator and the synthetic instances solved in this work can be found at https://github.com/alessioBonfietti/Multirate-Resource-Constraint-Repo.

[3] The time limit value refers to the limit we use for the subproblem of the last application (the hardest). The previous subproblems were solved with a time limit halved at each step (e.g. if we have three applications, the limits would be 75,150,300, respectively).

part includes more challenging instances, highlighting the performance of the approach in terms of time and solution quality. We have generated various sets of instances with 2,3, and 4 applications. Each set is characterized by (1) the number of applications, (2) the period multiplying factor, and (3) the graph structure of the instances (e.g. a set with 2 applications, factor 4, and parallel structure will be denoted as (2,4)par.). The performance are presented as the arithmetic average of the gaps of two metrics, where the gap is computed as follows: $Gap = 100 * \frac{X-M}{X}$ (where M is the value of the MRC approach and X is the value of the other one).

Table 1. Small synthetic instances results

Set	Struct.	Avg.# Act.	Mem Gap (%)		Tot.Time Gap(%)		Opt.Sol.Time (ms)		
			M vs. T&E	M vs. DJ	M vs. T&E	M vs. DJ	M	T&E	DJ
(2,2)	ser.	10.28	1.47 %	0.72 %	62.00 %	62.00 %	**0.00**	0.32	0.34
	par.	10.64	0.65 %	0.94 %	60.17 %	60.18 %	**0.02**	0.62	0.62
(2,4)	ser.	19.96	1.84 %	1.52 %	89.23 %	89.21 %	**0.86**	8.46	8.40
	par.	21.76	3.01 %	1.32 %	73.21 %	73.90 %	**28.90**	64.68	66.00
(3,2)	ser.	31.22	3.36 %	4.92 %	92.42 %	92.48 %	**0.26**	4.00	4.08
	par.	25.12	5.04 %	3.03 %	88.18 %	88.06 %	**0.24**	4.34	4.44
(4,2)	ser.	56.74	6.16 %	9.01 %	**94.71**%	**94.74**%	**47.84**	173.56	197.94
	par.	56.18	**10.23**%	7.23 %	88.52 %	88.47 %	**96.61**	419.28	475.87

Table 1 reports the results of the first experimental part. Each line corresponds to a set of 50 instances. The fourth and the fifth columns present the average memory consumption gap while the sixth and the seventh report the average total time gap (the total time includes the optimality proof). The columns from 8 to 10 show the average time spent finding the optimal solution (without the proof time). The third column reports the average number of activities. The performance of the proposed method increases as the instance size grows. The last line refers to the hardest experimental set of Table 1. Our approach requires up to 10 % less memory, computes the optimal solution with a speed up of over 4x and closes the search in nearly 10 % of the time. Note that our approach found and prove always the optimal solution in time, while both T&E and DJ where not able to prove the optimality in 4 instances (in the (4,2)par set). Figure 5 shows the performance profiles of the three approaches in the hardest sets (i.e.(4,2)ser and (4,2)par) of the first experimental part. Both graphics represent the percentage (y-axes) of optimal solutions found over time (x-axes). In the (4,2)ser set every approach concludes the search in less then 75 s, but note that we close the 88 % of the instances before 0.1 s (while T&E closes only the 30 % of the instances and DJ the 32 %). In the (4,2)par, both T&E and DJ close only the 92 % of the instances within the timelimit (300 s) while we close the 98 % of the instances before 75 s.

Table 2 reports the results of the second part of the synthetic experiments. The table reports the time gap spent in finding the first solution (which is
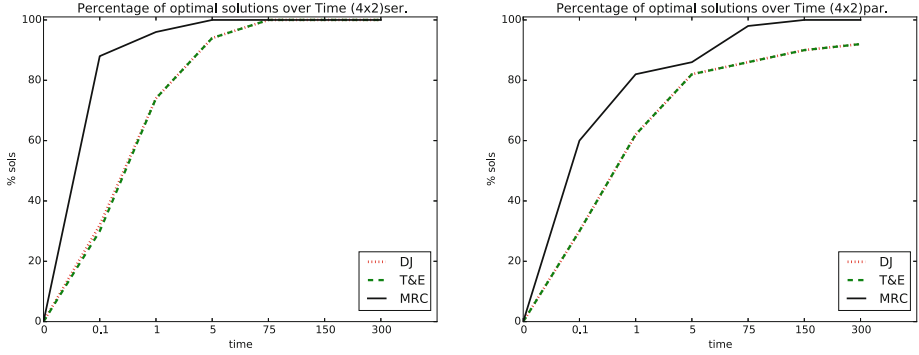
Fig. 5. Performance profiles of sets (4,2)seq. and (4,2)par.

obviously the same for all the approaches) and the solution value gap w.r.t. another approach called BaseLine. This approach is actually used in the industrial context when working with challenging instances. It consists in forcing all the activities of an application to be allocated on the same resource. The search is therefore very fast, but the solution lacks of quality. As reported in Table 2, our approach takes less then 1 % of the time w.r.t. T&E and DJ to find the first feasible solution solving challenging instances.

Table 2. Big synthetic instances results

			1st Sol.Time Gap (%)		Sol. Gap (%)
Set	Struct.	Avg.# Act.	M vs T&E	M vs DJ	vs BaseLine
(3,10)	ser.	525.13	99.172 %	99.179 %	57.66 %
	par.	544.40	99.301 %	99.304 %	57.50 %
(4,6)	ser.	1578.80	99.030 %	99.030 %	30.25 %
	par.	1419.93	99.408 %	99.410 %	43.23 %

Industrial Benchmarks. The real world instances come from the automation control system industry and were provided by ABB. Both instances are composed by three applications. The first instance, labelled *Real1*, consists of a 2353 activities and has been solved with a timelimit of 300 s. The second instance, labelled *Real2*, consists of 177646 activities and has been solved with a time limit of 2 h (i.e.7200 s). Table 3 reports some details of the instances (in the upper part) and the results of the experiments (in the bottom part). We recall that our approach, thanks to the multirate resource constraint, considers only a single repetition of each application, while any classical approach have to model each repetition as a variable. Hence, the MRC model of the *Real2* instance consists of **only** 205

interval variables, while both T&E and DJ models have 177646 variables. As a consequence our approach has a memory consumption less than 3 % w.r.t. to T&E and DJ and is able to find a solution in 159 ms, four orders of magnitude faster than T&E and DJ, which need 1827 and 2468 s, respectively. This has a huge impact over the search space explored, where the MRC approach within the time limit solved over 38 millions of branches, while T&E and DJ 2278 and 1522, respectively.

Table 3. Industrial instances and results

		Real1			Real2		
App 0	Period(μ-sec)	325000			50000		
	Act. Number	10			87		
App 1	Period(μ-sec)	1950000			2000000		
	Act. Number	332			72		
App 2	Period(μ-sec)	11700000			100000000		
	Act. Number	1			46		
Total	Act. Number	**2353**			**177646**		
Metric		M	T&E	DJ	M	T&E	DJ
Sol.Time(ms)		**5**	521	496	**159**	1827187	2468504
# Sol.		7	7	7	9	9	8
Mem (MB)		**14.9**	27.4	29.2	**34.4**	1258.3	1253.8
N branches		2	2	2	**38484544**	2278	1522
Sol.Gap vs Base(%)		50.85%	50.85%	50.85%	11.1%	11.1%	11.1%

6   Concluding Remarks

In this work we have presented a Constraint Programming approach for the multirate periodic scheduling problems. Key for the efficiency of the method is the ad-hoc search strategy and the multirate resource constraint which leverages on the modular algebra to enforce resource restrictions on the activities of the periodic applications. In the proposed algorithm the requirements and the resources are considered unary. We plan to investigate how to extend the algorithm considering cumulative resources.

References

1. Baptiste, P., Le Pape, C., Nuijten, W.: Constrains-Based Scheduling: Applying Constraint Programming to Scheduling. Springer, New York (2001)
2. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-Based Scheduling. Springer, New York (2001)
3. Beldiceanu, N., Carlsson, M., Thiel, S.: Sweep synchronization as a global propagation mechanism. Comput. Oper. Res. **33**(10), 2835–2851 (2006)
4. Shuvra, S., Bhattacharyya, S.S.: Embedded Multiprocessors - Scheduling and Synchronization. Signal Processing and Communications, 2nd edn. CRC Press, Boca Raton (2009)

5. Bonfietti, A., Lombardi, M., Benini, L., Milano, M.: Cross cyclic resource-constrained scheduling solver. Artif. Intell. **206**, 25–52 (2014)
6. Bonfietti, A., Lombardi, M., Milano, M.: De-cycling cyclic scheduling problems. In: Twenty-Third International Conference on Automated Planning and Scheduling (2013)
7. de Dinechin, B.D., Kordon, A.M.: Converging to periodic schedules for cyclic scheduling problems with resources and deadlines. Comput. Oper. Res. **51**, 227–236 (2014)
8. Laborie, P., Rogerie, J.: Reasoning with conditional time-intervals. In: Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15–17, 2008, Coconut Grove, Florida, USA, pp. 555–560 (2008)
9. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: Reasoning with conditional time-intervals. part II: an algebraical model for resources. In: Proceedings of the Twenty-Second International Florida Artificial Intelligence Research Society Conference, May 19–21, 2009, Sanibel Island, Florida, USA (2009)
10. Le Pape, C., Couronné, P., Vergamini, D., Gosselin, V.: Time-versus-capacity compromises in project scheduling. AISB QUARTERLY, p. 19 (1995)
11. Letort, A., Beldiceanu, N., Carlsson, M.: A scalable sweep algorithm. In: Proceedings of CP, pp. 439–454 (2012)
12. Li, Y., Wolf, W.: Hierarchical scheduling and allocation of multirate systems on heterogeneous multiprocessors. In: Proceedings of ED&TC, pp. 134–139 (1997)
13. Mercier, L., Van Hentenryck, P.: Edge finding for cumulative scheduling. INFORMS J. Comput. **20**(1), 143–153 (2008)
14. Google ortools. https://developers.google.com/optimization/
15. Vilim, P.: O(n log n) filtering algorithms for unary resource constraint. In: Proceedings of CPAIOR, pp. 335–347 (2004)
16. Yi, Y., Milward, M., Khawam, S., Nousias, L., Arslan, T.: Automatic synthesis and scheduling of multirate DSP algorithms. In: Proceedings of ASP-DAC, pp. 635–638 (2005)
17. Zhu, X.-Y., Basten, T., Geilen, M., Stuijk, S.: Efficient retiming of multirate DSP algorithms. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **31**(6), 831–844 (2012)
18. Zhu, X.-Y., Geilen, M., Basten, T., Stuijk, S.: Static rate-optimal scheduling of multirate DSP algorithms via retiming and unfolding. In: 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium, pp. 109–118. IEEE, April 2012

The Dichotomy for Conservative Constraint Satisfaction is Polynomially Decidable

Clément Carbonnel[1,2]([✉])

[1] CNRS, LAAS, 7 avenue du colonel Roche, 31400 Toulouse, France
carbonnel@laas.fr
[2] University of Toulouse, INP Toulouse, LAAS, 31400 Toulouse, France

Abstract. Given a fixed constraint language Γ, the conservative CSP over Γ (denoted by c-CSP(Γ)) is a variant of CSP(Γ) where the domain of each variable can be restricted arbitrarily. In [5] a dichotomy has been proven for conservative CSP: for every fixed language Γ, c-CSP(Γ) is either in P or NP-complete. However, the characterization of conservatively tractable languages is of algebraic nature and the recognition algorithm provided in [5] is super-exponential in the domain size. The main contribution of this paper is a polynomial-time algorithm that, given a constraint language Γ as input, decides if c-CSP(Γ) is tractable. In addition, if Γ is proven tractable the algorithm also outputs its *coloured graph*, which contains valuable information on the structure of Γ.

1 Introduction

The Constraint Satisfaction Problem (CSP) is a powerful framework for solving combinatorial problems, with many applications in artificial intelligence. A CSP instance is a set of variables, a set of values (the *domain*) and a set of constraints, which are relations imposed on a subset of variables. The goal is to assign to each variable a domain value in such a way that all constraints are satisfied. This problem is NP-complete in general.

A very active and fruitful research topic is the non-uniform CSP, in which a set of relations Γ is fixed and every constraint must be a relation from Γ. For instance, if Γ contains only binary Boolean relations then CSP(Γ) is equivalent to 2-SAT and hence polynomially solvable, but if all ternary clauses are allowed the problem becomes NP-complete. The Feder-Vardi Dichotomy Conjecture states that for every finite Γ, CSP(Γ) is either in P or NP-complete [10] (hence missing all the NP-intermediate complexity classes predicted by Ladner's Theorem [15]).

While this conjecture is still open, a major milestone was reached with the characterization of all tractable *conservative* constraint languages, that is, languages that contain every possible unary relation over their domain [5]. Conservativity is a very natural property since it corresponds to the languages that allow arbitrary restrictions of variables domains, a widely used feature in

practical constraint solving. It also includes as a particular case the well-studied problem List H-Colouring for a fixed digraph H.

Now that the criterion for the tractability of conservative languages has been established, an important question that arises is the complexity of *deciding* if a given conservative language is tractable. An algorithm that decides this criterion efficiently could be used for example as a preprocessing operation in general-purpose constraint solvers, and prompt the use of a dedicated algorithm instead of backtracking search if the instance is over a conservative tractable language.

This meta-problem can be phrased in two slightly different ways. The first would take the whole language Γ as input and ask if $\text{CSP}(\Gamma)$ is tractable. However, conservative languages always contain a number of unary relations that is exponential in the domain size, which inflates greatly the input size for the meta-problem without adding any computational difficulty. A more interesting question would take as input a language Γ and ask if c-$\text{CSP}(\Gamma)$ is tractable, where c-$\text{CSP}(\Gamma)$ allows all unary relations in addition to Γ (this is the *conservative CSP* over Γ). Designing a polynomial-time algorithm for this meta-problem is more challenging, but it would perform much better as a structural analysis tool for preprocessing CSP instances.

Bulatov's characterization of conservative tractable languages is based on the existence of closure operations (called *polymorphisms*) that satisfy a certain set of identities. While the algebraic nature of this criterion makes the meta-problem delicate to solve, it also shows that the meta-problem is in NP and can be solved in polynomial time if the domain size is fixed. This hypothesis is however very strong because there is only a finite number of constraint languages of fixed arity over a fixed domain. If the domain is not fixed this algorithm becomes super-exponential, and hence is polynomial for neither flavour of the meta-problem.

The contribution of our paper is twofold:

(i) We present an algorithm that decides the dichotomy for c-CSP in polynomial time. This is the main result of this paper.

(ii) As a byproduct, we exhibit a general connection between the complexity of the meta-problem and the existence of a *semiuniform algorithm* on classes of conservative languages defined by certain algebraic identities known as *linear strong Mal'tsev conditions*. We obtain as a corollary a broad generalization of the result about conservative Mal'tsev polymorphisms found in [7].

The necessary background for our proofs will be given in Sect. 2. In Sect. 3 we will then present the proof of the contribution (*ii*), and in Sect. 4 we will show how this result can be used to derive an algorithm that decides the dichotomy for c-CSP in polynomial time. Finally, we will conclude and discuss open problems in Sect. 5.

2 Preliminaries

2.1 Constraint Satisfaction Problems

An instance of the *constraint satisfaction problem* (CSP) is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ is a finite set of values and $\mathcal{C}$ is a set of constraints.

A *constraint* C of arity k is a pair (S_C, R_C) where R_C is a k-ary relation over $\mathcal{D}$ and $S_C \in \mathcal{X}^k$ is the *scope* of C. The goal is to find an assignment $\phi : \mathcal{X} \to \mathcal{D}$ such that for all $C = (S_C, R_C) \in \mathcal{C}$, $\phi(S_C) \in R_C$. In this definition, variables do not come with individual domains; any variable-specific domain restriction has to be enforced using a unary constraint.

Given a constraint $C = (S_C, R_C)$ and $X_1 \subseteq \mathcal{X}$, we denote by $C[X_1]$ the projection of C onto the variables in X_1 (which is the empty constraint if S does not contain any variable in X_1). The projection of a CSP instance I onto a subset $X_1 \subseteq \mathcal{X}$, denoted by $I_{|X_1}$, is obtained by projecting every constraint onto X_1 and then removing all variables that do not belong to X_1. A *partial solution* to I is a solution (i.e. a satisfying assignment) to $I_{|X_1}$ for some subset $X_1 \subseteq \mathcal{X}$. A CSP instance is *1-minimal* if each variable $x \in \mathcal{X}$ has an individual domain $D(x)$ (represented as a unary constraint) and the projection onto $\{x\}$ of every constraint $C \in \mathcal{C}$ whose scope contains x is exactly $D(x)$. 1-minimality can be enforced in polynomial time by gradually removing inconsistent tuples from the constraint relations until a fixed point is reached [16].

Throughout the paper we shall use $\mathcal{R}(.)$ and $\mathcal{S}(.)$ as operators that return respectively the relation and the scope of a constraint. A *constraint language* over a set $\mathcal{D}$ is a set of relations over $\mathcal{D}$, and the constraint language $\mathcal{L}(I)$ of a CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the set $\{\mathcal{R}(C) \mid C \in \mathcal{C}\}$. Given a constraint language Γ over a set $\mathcal{D}$, we denote by $\overline{\Gamma}$ the *conservative extension* of Γ, that is, the language comprised of Γ plus all possible unary relations over $\mathcal{D}$. Finally, given a constraint language Γ we denote by $\mathrm{CSP}(\Gamma)$ (resp. c-$\mathrm{CSP}(\Gamma)$) the restriction of CSP to instances I such that $\mathcal{L}(I) \subseteq \Gamma$ (resp. $\mathcal{L}(I) \subseteq \overline{\Gamma}$).

The algorithms presented in this paper will take constraint languages as input, and the complexity analysis depends crucially on how relations are encoded. While practical constraint solvers often represent relations intentionally through *propagators*, we shall always assume that every relation is given as an explicit list of tuples (a very common assumption in theoretical papers).

2.2 Polymorphisms

Given a constraint language Γ, the complexity of $\mathrm{CSP}(\Gamma)$ is usually studied through closure operations called polymorphisms. Given an integer k and a constraint language Γ over $\mathcal{D}$, a k-ary operation $f : \mathcal{D}^k \to \mathcal{D}$ is a *polymorphism* of Γ if for all $R \in \Gamma$ of arity r and $\mathbf{t_1}, \ldots, \mathbf{t_k} \in R$ we have

$$(f(\mathbf{t_1}[1], \ldots, \mathbf{t_k}[1]), \ldots, f(\mathbf{t_1}[r], \ldots, \mathbf{t_k}[r])) \in R$$

A polymorphism f is *idempotent* if $\forall x \in \mathcal{D}$, $f(x, \ldots, x) = x$ and *conservative* if $\forall x_1, \ldots, x_k \in \mathcal{D}$, $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$. It is known that given a constraint language Γ, the complexity of $\mathrm{CSP}(\Gamma)$ is entirely determined by its polymorphisms [13]. On the other hand, the conservative polymorphisms of Γ are exactly those that preserve all unary relations, and hence determine the complexity of c-$\mathrm{CSP}(\Gamma)$. A binary polymorphism f is a *semilattice* if $\forall x, y, z \in \mathcal{D}$, $f(x, x) = x$, $f(x, y) = f(y, x)$ and $f(f(x, y), z) = f(x, f(y, z))$.

A *majority* polymorphism is a ternary polymorphism f such that $\forall x, y \in \mathcal{D}$, $f(x,x,y) = f(x,y,x) = f(y,x,x) = x$ and a *minority* polymorphism is a ternary polymorphism f such that $\forall x, y \in \mathcal{D}$, $f(x,x,y) = f(x,y,x) = f(y,x,x) = y$.

2.3   Conservative Constraint Satisfaction

In general, if Γ is a conservative language and there exists $\{a,b\} \subseteq \mathcal{D}$ such that every polymorphism of Γ is a projection when restricted to $\{a,b\}$ then $\mathrm{CSP}(\{R\})$ is polynomially reducible to $\mathrm{CSP}(\Gamma)$ [14], where

$$R = \begin{pmatrix} a & b & b \\ b & a & b \\ b & b & a \end{pmatrix}$$

It follows that $\mathrm{CSP}(\Gamma)$ is NP-complete as $\mathrm{CSP}(\{R\})$ is equivalent to 1-in-3 SAT. The Dichotomy Theorem for conservative CSP states that the converse is true: if for every $B = \{a,b\} \subseteq \mathcal{D}$ there exists a polymorphism f such that $f_{|B}$ is *not* a projection, then c-$\mathrm{CSP}(\Gamma)$ is polynomial-time. By Post's lattice [17], the polymorphism f can be chosen such that $f_{|B}$ is either a majority operation, a minority operation or a semilattice.

Theorem 1 ([5]). *Let Γ be a fixed constraint language over a domain $\mathcal{D}$. If for every $B = \{a,b\} \subseteq \mathcal{D}$ there exists a conservative polymorphism f such that $f_{|B}$ is either a majority operation, a minority operation or a semilattice then c-$CSP(\Gamma)$ is in P. Otherwise, c-$CSP(\Gamma)$ is NP-complete.*

This theorem provides a way to determine the complexity of c-$\mathrm{CSP}(\Gamma)$, since we can enumerate all ternary operations over $\mathcal{D}$ and list those that are polymorphisms of Γ. However, this procedure is super-exponential in time if the domain is part of the input. Our paper presents a more elaborate, polynomial-time algorithm that does not impose any restriction on Γ.

Three different proofs of Theorem 1 have been published [1,5,6], and two of them rely heavily on a construction called the *coloured graph* of Γ and denoted by G_Γ. The definition of G_Γ is as follows. The vertex set of G_Γ is $\mathcal{D}$, and there is an edge between any two vertices. Each edge (a,b) is labelled with a colour following these rules:

– If there exists a polymorphism f such that $f_{|\{a,b\}}$ is a semilattice, then (a,b) is red;
– If there exists a polymorphism f such that $f_{|\{a,b\}}$ is a majority operation and (a,b) is not red, then (a,b) is yellow;
– If there exists a polymorphism f such that $f_{|\{a,b\}}$ is a minority operation and (a,b) is neither red nor yellow, then (a,b) is blue.

Additionally, red edges are directed: we have $(a \to b)$ if there exists f such that $f(a,b) = f(b,a) = b$. It is possible to have $(a \leftrightarrow b)$. By Theorem 1, G_Γ is entirely coloured if and only if c-$\mathrm{CSP}(\Gamma)$ is tractable. The next theorem, from [5], shows that the tractability of c-$\mathrm{CSP}(\Gamma)$ is always witnessed by three specific polymorphisms (instead of $O(d^2)$ in the original formulation).

Theorem 2 (The Three Operations Theorem [5]). *Let Γ be a language such that c-CSP(Γ) is tractable. There exist three conservative polymorphisms $f^*(x,y)$, $g^*(x,y,z)$ and $h^*(x,y,z)$ such that for every two-element set $B \subseteq \mathcal{D}$:*

- *$f^*_{|B}$ is a semilattice operation if B is red and $f^*(x,y) = x$ otherwise;*
- *$g^*_{|B}$ is a majority operation if B is yellow, $g^*_{|B}(x,y,z) = x$ if B is blue and $g^*_{|B}(x,y,z) = f^*(f^*(x,y),z)$ if B is red;*
- *$h^*_{|B}$ is a minority operation if B is blue, $h^*_{|B}(x,y,z) = x$ if B is yellow, and $h^*_{|B}(x,y,z) = f^*(f^*(x,y),z)$ if B is red.*

The original theorem also proves the existence of other polymorphisms, but we will only use f^*, g^* and h^* in our proofs.

2.4   Meta-Problems and Identities

Given a class T of constraint languages, the *meta-problem* (or *metaquestion* [8]) for T takes as input a constraint language Γ and asks if $\Gamma \in T$. In the context of CSP and c-CSP, the class T is often defined as the set of all languages that admit a combination of polymorphisms satisfying a certain set of identities; in this case the meta-problem is a *polymorphism detection problem*. We will be interested in particular sets of identities called *linear strong Mal'tsev conditions*. Given that universal algebra is not the main topic of our paper, we will use a simplified exposition similar to that found in [8]. A *linear identity* is an expression of the form $f(x_1, \ldots, x_k) \approx g(y_1, \ldots, y_c)$ or $f(x_1, \ldots, x_k) \approx y_i$ where f, g are operation symbols and $x_1, \ldots, x_k, y_1, \ldots, y_c$ are variables. It is *satisfied* by two interpretations for f and g on a domain $\mathcal{D}$ if the equality holds for any assignment to the variables. A *strong linear Mal'tsev condition* $\mathcal{M}$ is a finite set of linear identities. We say that a set of operations satisfy $\mathcal{M}$ if they satisfy every identity in $\mathcal{M}$. A strong linear Mal'tsev condition is said to be *idempotent* if it entails $f_i(x, \ldots, x) \approx x$ for all operation symbols f_i. For a given linear strong Mal'tsev condition, the number of operation symbols and their maximum arity are constant.

Example 1. The set of identities

$$f(x,x,y) \approx x$$
$$f(x,y,x) \approx x$$
$$f(y,x,x) \approx x$$

is the idempotent linear strong Mal'tsev condition that defines majority operations. On the other hand, recall that semilattices are binary operations f satisfying

$$f(x,x) \approx x$$
$$f(x,y) \approx f(y,x)$$
$$f(x, f(y,z)) \approx f(f(x,y),z)$$

which does not form a linear strong Mal'tsev condition because the identity enforcing the associativity of f is not linear.

By extension, we say that a constraint language satisfies a linear strong Mal'tsev condition $\mathcal{M}$ if it has a collection of polymorphisms that satisfy $\mathcal{M}$. The definability of a class of constraint languages by a linear strong Mal'tsev condition $\mathcal{M}$ is strongly tied up with the meta-problem, because for such classes we can associate any constraint language Γ with a polynomial-sized CSP instance whose solutions, if any, are exactly the polymorphisms of Γ satisfying $\mathcal{M}$ [8]. We will describe the construction below.

Given a constraint language Γ and an integer k the *indicator problem* of order k of Γ, denoted by $\mathcal{IP}^k(\Gamma)$, is a CSP instance with one variable $x_{f(d_1,\ldots,d_k)}$ for every $(d_1,\ldots,d_k) \in \mathcal{D}^k$ and one constraint $C^{R^*}_{\mathbf{f}(\mathbf{t_1},\ldots,\mathbf{t_k})}$ for each $R^* \in \Gamma$, $\mathbf{t_1},\ldots,\mathbf{t_k} \in R^*$. The constraint $C^{R^*}_{\mathbf{f}(\mathbf{t_1},\ldots,\mathbf{t_k})}$ has R^* as relation, and its scope S is such that for all $i \leq |S|$, $S[i] = x_{f(\mathbf{t_1}[i],\ldots,\mathbf{t_k}[i])}$. Going back to the definition of a polymorphism, it is simple to see that the solutions to $\mathcal{IP}^k(\Gamma)$ are exactly the k-ary polymorphisms of Γ [13].

Now, let $\mathcal{M}$ denote a linear strong Mal'tsev condition with symbols $f_1,\ldots,f_m$ of respective arities $a_1,\ldots,a_m$. We build a CSP instance $\mathcal{P}_\mathcal{M}(\Gamma)$ that is the disjoint union of $\mathcal{IP}^{a_1}(\Gamma),\ldots,\mathcal{IP}^{a_m}(\Gamma)$. By construction, each solution ϕ to $\mathcal{P}_\mathcal{M}(\Gamma)$ is a collection of polymorphisms $(f_1,\ldots,f_m)$ of Γ. We can force these polymorphisms to satisfy the identities in $\mathcal{M}$ by adding new constraints. If $\mathcal{E}_i \in \mathcal{M}$ is of the form $f_j(x_1,\ldots,x_{a_j}) \approx f_p(y_1,\ldots,y_{a_p})$, we add an equality constraint between the variables $x_{f_j(\phi(x_1),\ldots,\phi(x_{a_j}))}$ and $x_{f_p(\phi(y_1),\ldots,\phi(y_{a_p}))}$ for every possible assignment ϕ to $\{x_1,\ldots,x_{a_j},y_1,\ldots,y_{a_p}\}$. Otherwise (i.e. if $\mathcal{E}_i$ is of the form $f_j(x_1,\ldots,x_k) \approx y_i$) we can enforce $\mathcal{E}_i$ by adding unary constraints. Note that the language of $\mathcal{P}_\mathcal{M}(\Gamma)$ is Γ together with possible equalities and unary relations with a single tuple. This construction will be used frequently throughout the paper.

2.5 Uniform and Semiuniform Algorithms

Let $\mathcal{M}$ denote a strong linear Mal'tsev condition, and let $\mathrm{CSP}(\mathcal{M})$ denote the CSP restricted to instances whose language satisfies $\mathcal{M}$.

Definition 1. *A* uniform polynomial-time algorithm *for $\mathcal{M}$ is an algorithm that solves CSP($\mathcal{M}$) in polynomial time.*

The term "uniform" here refers to the fact that the language is not fixed (as in the Feder-Vardi Dichotomy conjecture), but may only range over languages that satisfy $\mathcal{M}$. The existence of a uniform algorithm implies that $\mathrm{CSP}(\Gamma)$ is in P for every Γ that satisfies $\mathcal{M}$, but the converse is not guaranteed to be true. For instance, an algorithm for $\mathrm{CSP}(\mathcal{M})$ that is exponential only in the domain size is polynomial for every fixed Γ that satisfies $\mathcal{M}$, but is not uniform. A weaker notion of uniformity called *semiuniformity* has been recently introduced in [8], and will be central to our paper.

Definition 2. *A semiuniform polynomial-time algorithm for $\mathcal{M}$ is an algorithm that solves CSP($\mathcal{M}$) in polynomial time provided each instance I is paired with polymorphisms $f_1, \ldots, f_m$ of $\mathcal{L}(I)$ that satisfy $\mathcal{M}$.*

Observe that semiuniform algorithms are tied to the identities in $\mathcal{M}$ rather than the class of languages it defines; even if CSP($\mathcal{M}_1$) and CSP($\mathcal{M}_2$) denote the exact same set of instances, the polymorphisms satisfying $\mathcal{M}_2$ can be more computationally useful than those satisfying $\mathcal{M}_1$.

The following observation has been part of the folklore for some time (see e.g. [2,4]) and has been recently formalized in [8].

Proposition 1 ([8]). *Let $\mathcal{M}$ be an idempotent strong linear Mal'tsev condition. If $\mathcal{M}$ has a uniform algorithm, then the meta-problem for $\mathcal{M}$ is polynomial time.*

We give here the proof sketch. The idempotency of $\mathcal{M}$ ensures that we have a uniform algorithm for the *search* problem (i.e. decide if the instance is satisfiable and produce a solution if one exists) because idempotent polymorphisms always preserve assignments to variables, which can be seen as unary relations with a single tuple. Given a relational structure Γ, to check if Γ satisfies $\mathcal{M}$ we build the instance $\mathcal{P}_{\mathcal{M}}(\Gamma)$ as in Sect. 2.4 and invoke the uniform search algorithm. Since the language of $\mathcal{P}_{\mathcal{M}}(\Gamma)$ is Γ plus equalities and unary relations with a single tuple, $\mathcal{L}(\mathcal{P}_{\mathcal{M}}(\Gamma))$ satisfies $\mathcal{M}$ if and only if Γ does. If $\mathcal{P}_{\mathcal{M}}(\Gamma)$ is satisfiable then Γ satisfies $\mathcal{M}$ and the algorithm must produce a solution (which can be easily verified), and whenever the algorithm fails to do so we can safely conclude that Γ does not satisfy $\mathcal{M}$.

There is no intuitive way to make this approach work with semiuniform algorithms because they will not run unless given an explicit solution to $\mathcal{P}_{\mathcal{M}}(\Gamma)$ beforehand.

3   Semiuniformity in Conservative Constraint Languages

As seen in Sect. 2.5, in the case of idempotent linear strong Mal'tsev conditions a uniform algorithm implies the tractability of the meta-problem. We will see that if the problem is to decide if $\overline{\Gamma}$ satisfies $\mathcal{M}$ (i.e. to decide if Γ has *conservative* polymorphisms $f_1, \ldots, f_m$ that satisfy $\mathcal{M}$) then semiuniformity is sufficient. This implies that, surprisingly, *uniformity and semiuniformity are equivalent* for classes of conservative languages definable by a strong linear Mal'tsev condition.

The general strategy to solve the meta-problem assuming a semiuniform algorithm is to cast the meta-problem as a CSP and then compute successively partial solutions $\phi_1, \ldots, \phi_\alpha$ of slowly increasing size until a solution to the whole CSP is obtained. The originality of our approach is that ϕ_{i+1} is not computed directly from ϕ_i, but by solving a polynomial number of CSP instances whose languages admit ϕ_i as a polymorphism. This algorithm can be seen as a treasure hunt, where each chest contains the key to open the next one.

Let $\mathcal{M}$ be a strong linear Mal'tsev condition with operation symbols $f_1, \ldots, f_m$ of respective arities $a_1, \ldots, a_m$. Let Γ be a constraint language over

$\mathcal{D}$ and $\mathcal{P}_{\mathcal{M}}(\Gamma)$ be the CSP whose solutions are exactly the polymorphisms of Γ satisfying $\mathcal{M}$ (as described in Sect. 2.4). Recall that for every symbol f_i in $\mathcal{M}$ and $(d_1, \ldots, d_{a_i}) \in \mathcal{D}^{a_i}$ we have a variable $x_{f_i(d_1,\ldots,d_{a_i})}$ that dictates how f_i should map $d_1, \ldots, d_{a_i}$, and for every $R^* \in \Gamma$ and a_i tuples $\mathbf{t_1}, \ldots, \mathbf{t_{a_i}} \in R^*$ we have a constraint $C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}}$ that forces the tuple $\mathbf{f_i(t_1, \ldots, t_{a_i})}$ to belong to R^* (where $\mathbf{f_i}$ is the operation on tuples obtained by componentwise application of f_i). Our goal is to decide if $\overline{\Gamma}$ satisfies $\mathcal{M}$, which requires the polymorphisms of Γ satisfying $\mathcal{M}$ to be conservative. The solutions to $\mathcal{P}_{\mathcal{M}}(\Gamma)$ can easily be guaranteed to be conservative by adding the unary constraint $x_{f_i(d_1,\ldots,d_{a_i})} \in \{d_1, \ldots, d_{a_i}\}$ on each variable $x_{f_i(d_1,\ldots,d_{a_i})} \in \mathcal{X}$. We will denote this new problem by $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, and each solution ϕ to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ is a collection $(f_1, \ldots, f_m)$ of conservative polymorphisms of Γ satisfying $\mathcal{M}$.

We need one more definition. Given a CSP instance $\mathcal{I}$, a *consistent restriction* of $\mathcal{I}$ is an instance obtained from $\mathcal{I}$ by adding new constraints that are either unary or equalities and then enforcing 1-minimality. We will be interested in the consistent restrictions of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, and we will keep the same notations for constraints that already existed in $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. The next lemma is a variation of ([7], Observation 2) adapted to our purpose.

Lemma 1. *Let $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a consistent restriction of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. Let f_i and f_j be operation symbols in $\mathcal{M}$. If $C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}} \in \mathcal{C}$ and $\mathbf{t'_1}, \ldots, \mathbf{t'_{a_j}} \in \mathcal{R}(C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}})$ then*

$$\mathcal{R}(C^{R^*}_{\mathbf{f_j(t'_1,\ldots,t'_{a_j})}}) \subseteq \mathcal{R}(C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}})$$

Proof. Let $S = \mathcal{S}(C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}})$ and $S' = \mathcal{S}(C^{R^*}_{\mathbf{f_j(t'_1,\ldots,t'_{a_j})}})$. Before 1-minimality was enforced, we had $\mathcal{R}(C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}}) = \mathcal{R}(C^{R^*}_{\mathbf{f_j(t'_1,\ldots,t'_{a_j})}}) = R^*$. Thus, after enforcing 1-minimality we have $\mathcal{R}(C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}}) = R^* \cap (\pi_{x \in S} D(x))$ and $\mathcal{R}(C^{R^*}_{\mathbf{f_j(t'_1,\ldots,t'_{a_j})}}) = R^* \cap (\pi_{x \in S'} D(x))$. However, since $\mathbf{t'_1}, \ldots, \mathbf{t'_{a_j}} \in \mathcal{R}(C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}})$, the conservativity constraints ensure that for each k,

$$D(S'[k]) = D(x_{f_j(\mathbf{t'_1}[k],\ldots,\mathbf{t'_{a_j}}[k])}) \subseteq \{\mathbf{t'_1}[k], \ldots, \mathbf{t'_{a_j}}[k]\} \subseteq D(S[k])$$

Therefore, $\mathcal{R}(C^{R^*}_{\mathbf{f_j(t'_1,\ldots,t'_{a_j})}}) \subseteq \mathcal{R}(C^{R^*}_{\mathbf{f_i(t_1,\ldots,t_{a_i})}})$.

Given two sets of variables $X_1, X_2 \subseteq \mathcal{X}$, we write $X_1 \lhd X_2$ if for each symbol f_i in $\mathcal{M}$, $\forall x \in X_2$ and $\mathbf{t} \in D(x)^{a_i}$ we have $x_{f_i(\mathbf{t})} \in X_1$. If $X_1 \lhd X_1$, we say that X_1 is *closed*.

Proposition 2. *Let $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a consistent restriction of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. If X_1 and X_2 are subsets of variables such that $X_1 \lhd X_2$, then every solution to $\mathcal{P}_{|X_1}$ is a collection of polymorphisms of $\mathcal{L}(\mathcal{P}_{|X_2})$.*

Proof. Let $f_i, f_j \in \{f_1, \ldots, f_m\}$ be operation symbols in $\mathcal{M}$. Let $R^* \in \Gamma$, $\mathbf{t_1}, \ldots, \mathbf{t_{a_i}} \in R^*$, $C_2 = (S_2, R_2) \in \mathcal{P}_{|X_2}$ be the projection of $C_{\mathbf{f_i}(\mathbf{t_1}, \ldots, \mathbf{t_{a_i}})}^{R^*}$ onto X_2, and $\mathbf{t_1^2}, \ldots, \mathbf{t_{a_j}^2} \in R_2$. By the nature of projections, there must exist $\mathbf{t_1'}, \ldots, \mathbf{t_{a_j}'} \in \mathcal{R}(C_{\mathbf{f_i}(\mathbf{t_1}, \ldots, \mathbf{t_{a_i}})}^{R^*})$ such that $\mathbf{t_1^2}, \ldots, \mathbf{t_{a_j}^2}$ is the projection of $\mathbf{t_1'}, \ldots, \mathbf{t_{a_j}'}$ onto X_2. Then, by Lemma 1 we have

$$\mathcal{R}(C_{\mathbf{f_j}(\mathbf{t_1'}, \ldots, \mathbf{t_{a_j}'})}^{R^*}) \subseteq \mathcal{R}(C_{\mathbf{f_i}(\mathbf{t_1}, \ldots, \mathbf{t_{a_i}})}^{R^*})$$

and in particular $\mathcal{R}(C_{\mathbf{f_j}(\mathbf{t_1'}, \ldots, \mathbf{t_{a_j}'})}^{R^*}[X_2]) \subseteq \mathcal{R}(C_{\mathbf{f_i}(\mathbf{t_1}, \ldots, \mathbf{t_{a_i}})}^{R^*}[X_2]) = R_2$. Now, note that because $X_1 \lhd X_2$ and $\mathcal{P}$ is 1-minimal, every variable $x_{f_j(\mathbf{t_1'}[k], \ldots, \mathbf{t_{a_j}'}[k])}$ in the scope of $C_{\mathbf{f_j}(\mathbf{t_1'}, \ldots, \mathbf{t_{a_j}'})}^{R^*}[X_2]$ also belongs to X_1. We denote this constraint by C_1.

Let us summarize what we have: for every symbol f_j, every relation $R_2 \in \mathcal{L}(\mathcal{P}_{|X_2})$ other than equalities and unary relations (which are preserved by all conservative polymorphisms) and $\mathbf{t_1^2}, \ldots, \mathbf{t_{a_j}^2} \in R_2$, there is a constraint $C_1 = (S_1, R_1) \in \mathcal{P}_{|X_1}$ such that $|S_1| = |S_2|$, $R_1 \subseteq R_2$ and for every k we have $S_1[k] = x_{f_j(\mathbf{t_1^2}[k], \ldots, \mathbf{t_{a_j}^2}[k])}$. It follows that for every solution $(f_1, \ldots, f_m)$ to $\mathcal{P}(\Gamma)_{|X_1}$, f_j is also a solution to the indicator problem of order a_j of $\mathcal{L}(\mathcal{P}(\Gamma)_{|X_2})$ and is therefore a polymorphism of $\mathcal{L}(\mathcal{P}(\Gamma)_{|X_2})$.

Closed sets of variables allow us to turn partial solutions into true polymorphisms of a specific constraint language, hence enabling us to make (limited) use of semiuniform algorithms. A variable of $\mathcal{P}_{\mathcal{M}}^c(\Gamma)$ is a *singleton* if it is of the form $x_{f_i(v, \ldots, v)}$ for some $v \in \mathcal{D}$. The sets of variables corresponding to singletons and $\mathcal{X}$ constitute two closed sets; the next Lemma shows that many intermediate, regularly-spaced closed sets exist in $\mathcal{P}_{\mathcal{M}}^c(\Gamma)$ between these two extremes.

Lemma 2. *Let $\mathcal{P}_{\mathcal{M}}^c(\Gamma) = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ after applying 1-minimality. There exist $X_0 \subseteq \ldots \subseteq X_\alpha = \mathcal{X}$ such that X_0 is the set of all singleton variables, each X_i is closed and $|X_{i+1} - X_i| \leq ma^a$, where a and m denote respectively the maximum arity and number of operation symbols in $\mathcal{M}$.*

Proof. Let $(D_1, \ldots, D_\alpha)$ denote an arbitrary ordering of the subsets of $\mathcal{D}$ of size a. We define

$$X_0 = \{x_{f_j(v_i, \ldots, v_i)} \mid f_j \in \mathcal{M}, v_i \in \mathcal{D}\}$$

and for all $i \in [1..\alpha]$

$$X_i = X_{i-1} \cup \{x_{f_j(\mathbf{t})} \mid f_j \in \mathcal{M}, \mathbf{t} \in (D_i)^{a_j}\}$$

It is clear that X_0 is the set of all singleton variables and for all i, $|X_{i+1} - X_i| \leq m|(D_i)^a| = ma^a$. It remains to show that each set is closed. Let $k \geq 1$ and suppose that X_{k-1} is closed. By induction hypothesis, we only need to verify that $X_k \lhd X_k \backslash X_{k-1}$. Let $x_{f_j(v_1, \ldots, v_{a_j})}$ be a variable in $X_k \backslash X_{k-1}$. Because $\mathcal{P}_{\mathcal{M}}^c(\Gamma)$ is 1-minimal, we have $D(x_{f_j(v_1, \ldots, v_{a_j})}) \subseteq \{v_1, \ldots, v_{a_j}\} \subseteq D_k$. By construction X_k contains all variables of the form $x_{f_c(\mathbf{t})}$ where $\mathbf{t} \in (D_k)^{a_c}$

and because $D(x_{f_j(v_1,\ldots,v_{a_j})}) \subseteq \{v_1,\ldots,v_{a_j}\} \subseteq D_k$ it contains in particular all variables $x_{f_c(\mathbf{t})}$ such that $t \subseteq D(x_{f_j(v_1,\ldots,v_{a_j})})$. This implies that $X_k \lhd X_k \backslash X_{k-1}$ and concludes the proof.

We now have every necessary tool at our disposal to start solving $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. It is straightforward to see that if a subset of variables X' is closed in $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, then it is closed in every consistent restriction as well.

Proposition 3. *If a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_i}$ is known, then a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_{i+1}}$ can be found in polynomial time.*

Proof. Let $(f^i_1,\ldots,f^i_m)$ be a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_i}$. We assume that 1-minimality has been enforced on $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. This ensures, in particular, that the domain of each $x_{f_j(\mathbf{t})} \in X_{i+1}\backslash X_i$ contains at most a elements. It follows that $X_{i+1}\backslash X_i$ has at most $s = a^{ma^a}$ possible assignments $\phi_1,\ldots,\phi_s$. For every $j \in [1..s]$, we create a CSP instance $\mathcal{P}_j$ that is a copy of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ but also includes the constraints corresponding to the assignment $X_{i+1}\backslash X_i \leftarrow \phi_j(X_{i+1}\backslash X_i)$. We enforce 1-minimality on every instance $\mathcal{P}_j$.

Now, observe that each $\mathcal{P}_j$ is a consistent restriction of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, so X_i is still closed in $\mathcal{P}_j$. Moreover, every variable $x \in X_{i+1}\backslash X_i$ has domain size 1 in $\mathcal{P}_j$; since X_i contains all singleton variables, if follows that in $\mathcal{P}_j$ we have $X_i \lhd X_{i+1}$.

By Proposition 2, $(f^i_1,\ldots,f^i_m)$ is a collection of polymorphisms of $\mathcal{L}(\mathcal{P}_{j|X_{i+1}})$. We can then use the semiuniform algorithm to find in polynomial time a solution to $\mathcal{P}_{j|X_{i+1}}$ if one exists by backtracking search (every f^i_z is idempotent, so we can invoke the semiuniform algorithm at each node to ensure that the algorithm cannot backtrack more than one level). A solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_{i+1}}$ exists if and only if $\mathcal{P}_{j|X_{i+1}}$ has a solution for some $j \in \{1,\ldots,s\}$.

The above proof balances on the fact that every complete instantiation of the variables in $X_{i+1}\backslash X_i$ (followed by 1-minimality) yields a residual instance over a language that admits $(f^i_1,\ldots,f^i_m)$ as polymorphisms. In other terms, $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_{i+1}}$ has a *backdoor* [19] of constant size to $(f^i_1,\ldots,f^i_m)$.

Theorem 3. *Let $\mathcal{M}$ be a linear strong Mal'tsev condition that admits a semiuniform algorithm. There exists a polynomial-time algorithm that, given as input a constraint language Γ, decides if $\overline{\Gamma}$ satisfies $\mathcal{M}$ and produces conservative polymorphisms of Γ satisfying $\mathcal{M}$ if any exist.*

Proof. The algorithm starts by building $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ and computes the sets $X_0,\ldots,X_\alpha$ as in Lemma 2. We have a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_0}$ for free because of the conservativity constraints, and we can compute a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ by invoking repeatedly (at most $\alpha \leq |\mathcal{X}| \leq md^a$ times) Proposition 3.

Corollary 1. *If $\mathcal{M}$ is a linear strong Mal'tsev condition that has a semiuniform algorithm for conservative languages, then $\mathcal{M}$ has also a uniform algorithm for conservative languages.*

Proof. The uniform algorithm simply invokes our algorithm to produce the conservative polymorphisms satisfying $\mathcal{M}$, and then provides these polymorphisms to the semiuniform algorithm to solve the CSP instance.

An immediate application of Theorem 3 concerns the detection of conservative k-edge polymorphisms for a fixed k. A k-edge operation on a set $\mathcal{D}$ is a $(k+1)$-ary operation e satisfying

$$e(x,x,y,y,y,\ldots,y,y) \approx y$$
$$e(x,y,x,y,y,\ldots,y,y) \approx y$$
$$e(x,y,y,x,y,\ldots,y,y) \approx y$$
$$e(x,y,y,y,x,\ldots,y,y) \approx y$$
$$\ldots$$
$$e(x,y,y,y,y,\ldots,x,y) \approx y$$
$$e(x,y,y,y,y,\ldots,y,x) \approx y$$

These identities form a linear strong Mal'tsev condition. The algorithm given in [12] is semiuniform, but in addition to e it must have access to three other polymorphisms p,d,s derived from e and satisfying

$$p(x,y,y) \approx x$$
$$p(x,x,y) \approx d(x,y)$$
$$d(x,d(x,y)) \approx d(x,y)$$
$$s(x,y,y,y,\ldots,y,y) \approx d(y,x)$$
$$s(y,x,y,y,\ldots,y,y) \approx y$$
$$s(y,y,x,y,\ldots,y,y) \approx y$$
$$\ldots$$
$$s(y,y,y,y,\ldots,y,x) \approx y$$

The authors provide a method to obtain these three polymorphisms from e that requires a possibly exponential number of compositions. However, conservative algebras are much simpler and we can observe that

$$s(x_1,x_2,\ldots,x_k) = e(x_2,x_1,x_2,x_3,\ldots,x_k)$$
$$d(x,y) = e(x,y,x,\ldots,x)$$
$$p(x,y,z) = e(y,d(y,z),x,\ldots,x)$$

satisfy the required identities and are easy to compute. It follows that in the conservative case their algorithm is semiuniform even if only a k-edge polymorphism e is given.

Corollary 2. *For every fixed k, the class of constraint languages admitting a conservative k-edge polymorphism is uniformly tractable and has a polynomially decidable meta-problem.*

Since conservative 2-edge polymorphisms are Mal'tsev polymorphisms, this corollary is a broad generalization of the result obtained in [7] concerning conservative Mal'tsev polymorphisms.

4 Deciding the Dichotomy

While the criterion for the conservative dichotomy theorem can be stated as a linear strong Mal'tsev condition [18], none of the algorithms found in the literature are semiuniform. Still, Theorem 3 gives a uniform algorithm for constraint languages Γ whose coloured graph contains only yellow and blue edges: if $g^*(x, y, z)$ and $h^*(x, y, z)$ are the polymorphisms predicted by the Three Operations Theorem, then $m^*(x, y, z) = h^*(g^*(x, y, z), g^*(y, z, x), g^*(z, x, y))$ is a generalized majority-minority polymorphism of Γ (see [9] for a formal definition), which implies that Γ has a 3-edge polymorphism [3].

Our algorithm will reduce the meta-problem to a polynomial number of CSP instances over languages with conservative 3-edge polymorphisms using a refined version of the treasure hunt algorithm and a simple reduction rule. This reduction rule is specific to indicator problems and allows us to avoid the elaborate machinery presented in [6] to eliminate red edges in CSP instances over a tractable conservative language.

We start by the reduction rule. Recall that the Three Operations Theorem predicts that if Γ is tractable then it has a conservative polymorphism f^* such that for every 2-element set B, $f^*_{|B}$ is a semilattice if B is red and $f^*_{|B}(x, y) = x$ otherwise.

Proposition 4. *If f^* is known, then for every non-red 2-element subset B of $\mathcal{D}$ it can be decided in polynomial time if there exists a conservative polymorphism p such that $p_{|B}$ is a majority (resp. minority) operation.*

Proof. We are looking for a ternary polymorphism p, so we start by building the instance $\mathcal{IP}^{3c}(\Gamma)$, which is the indicator problem of order 3 of Γ with conservativity constraints. For $i \in \{1, 2, 3\}$, let π_i be the solution to $\mathcal{IP}^{3c}(\Gamma)$ given by $\pi_i(x_{v_1, v_2, v_3}) = v_i$ for all $v_1, v_2, v_3 \in \mathcal{D}$. These solutions correspond to the three ternary polymorphisms of Γ that project onto their ith argument. We enforce 1-minimality and apply the algorithm Reduce.

We denote by $\mathcal{IP}^{3c}_R(\Gamma)$ the resulting CSP instance. An important invariant of this algorithm is that at the end of every iteration of the loop in Reduce, for every $x \in \mathcal{X}$ and $v \in D(x)$ there exists $s \in \{s_1, s_2, s_3\}$ such that $s(x) = v$. This is straightforward, since we only remove v from $D(x)$ if none of $s_1(x), s_2(x), s_3(x)$ takes value v. It then follows from the loop condition that at the end of Reduce, no $x \in \mathcal{X}$ may have a domain that contains a red pair of elements.

We now show that if $\mathcal{IP}^{3c}(\Gamma)$ has a solution that is majority (resp. minority) on a non-red pair of values B, then so does $\mathcal{IP}^{3c}_R(\Gamma)$. We proceed by induction. Suppose that at iteration i of the loop of Reduce, a solution p_i that is majority (resp. minority) on B exists. Let $D_i(x)$ denote the domain of a variable x at step i. We set $p_{i+1} = f^*(p_i, s_j)$. Because f always projects onto its first argument

---

Algorithm 1. Reduce

---

1 $s_1 \leftarrow \pi_1$;
2 $s_2 \leftarrow \pi_2$;
3 $s_3 \leftarrow \pi_3$;
4 **while** *There exist* i, j *and* $x \in \mathcal{X}$ *such that* $\{s_i(x), s_j(x)\}$ *is red and*
     $f^*(s_i(x), s_j(x)) = s_j(x)$ **do**
5 $\quad s_1 \leftarrow f^*(s_1, s_j)$;
6 $\quad s_2 \leftarrow f^*(s_2, s_j)$;
7 $\quad s_3 \leftarrow f^*(s_3, s_j)$;
8 $\quad$ **for** *all* $x \in \mathcal{X}$ *and* $v \in D(x)$ *s.t.* $\forall k, s_k(x) \neq v$ **do**
9 $\quad\quad D(x) \leftarrow D(x)\backslash v$;

---

on non-red pairs, a value v can only be removed from $D_i(x)$ at iteration $i+1$ if $\{v, s_j(x)\}$ is red and $f(v, s_j(x)) = s_j(x)$. Therefore, if $p_i(x)$ is removed at iteration i then $p_{i+1}(x) = f^*(p_i(x), s_j(x)) = s_j(x)$, and otherwise $p_{i+1}(x) \in \{p_i(x), s_j(x)\} \subseteq D_{i+1}(x)$; in any case $p_{i+1}(x) \in D_{i+1}(x)$. Furthermore, since B is not red, $p_{i+1}(x_{f(v_1,v_2,v_3)}) = p_i(x_{f(v_1,v_2,v_3)})$ for all $\{v_1, v_2, v_3\} \subseteq B$ and we can conclude that p_{i+1} is still majority (resp. minority) on B.

Now, we enforce 1-minimality again. We can ensure that every solution is a majority (resp. minority) polymorphism when restricted to B by assigning the 6 variables concerned by the majority (resp. minority) identity. Since the remaining instance I is red-free in G_Γ, either c-CSP(Γ) is intractable or $\mathcal{L}(I)$ admits a 3-edge polymorphism. We test for the existence of a 3-edge polymorphism using Theorem 3. If one exists we use the uniform algorithm given by Corollary 2 to decide if a solution exists and otherwise we can conclude that c-CSP(Γ) is intractable.

With this result in mind, the last challenge is to design a polynomial-time algorithm that finds a binary polymorphism f^* that is commutative on as many 2-element subsets as possible, and projects onto its first argument otherwise. We call such polymorphisms *maximally commutative*. This can be achieved using a variant of the algorithm presented in Sect. 3 and the following Lemma.

Lemma 3. *Let* $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ *denote an 1-minimal instance such that* $\forall x \in \mathcal{X}$, $|D(x)| \leq 2$. *Suppose that we have a conservative binary polymorphism* f *of* $\mathcal{L}(\mathcal{P})$ *and a partition* (V_1, V_2) *of the variables such that* $f(a, b) = f(b, a) = f(D(x))$ *whenever* $x \in V_1$, *and* f *projects onto its first argument otherwise. Then, every variable* $x \in V_1$ *can be assigned to* $f(D(x))$ *without altering the satisfiability of* $\mathcal{P}$.

Proof. Let $C = (S, R) \in \mathcal{C}$. Let $S_1 = S \cap V_1$, $S_2 = S \cap V_2$ and $\mathbf{t} \in R$. We assume without loss of generality that no variable in S is ground (i.e. has a singleton domain). If $x \in S$, let $\overline{\mathbf{t}[x]} = D(x)\backslash\mathbf{t}[x]$. Because $\mathcal{P}$ is 1-minimal, for every $x \in S_1$

there exists $\mathbf{t_x} \in R$ such that $\mathbf{t_x}[x] = \overline{\mathbf{t}[x]}$. Let $x_1, \ldots, x_s$ denote an arbitrary ordering of S_1. Then, let $\mathbf{t^{(0)}} = \mathbf{t}$ and for $i \in \{1, \ldots, s\}$,

$$\mathbf{t^{(i)}} = \mathbf{f}(\mathbf{t^{(i-1)}}, \mathbf{t_{x_i}})$$

It is immediate to see that if $x \in S_2$, then $\mathbf{t^{(s)}}[x] = \mathbf{t}[x]$ since f will project onto its first argument at each interation. On the other hand, if $x_k \in S_1$ and there exists j such that $\mathbf{t^{(j)}}[x_k] = f(D(x_k))$ then $\mathbf{t^{(i)}}[x_k] = f(D(x_k))$ for all $i \geq j$. This is guaranteed to happen for $j \leq k$, as either

- $\mathbf{t}[x_k] = f(D(x_k))$, in which case it is true for $j = 0$, or
- $\mathbf{t^{(k-1)}}[x_k] = f(D(x_k))$, in which case it is true for $j = k - 1$, or
- $\mathbf{t^{(k-1)}}[x_k] = \mathbf{t}[x_k] \neq f(D(x_k))$, in which case $\mathbf{t^{(k)}}[x_k] = f(\mathbf{t^{(k-1)}}[x_k],$ $\mathbf{t_{x_k}}[x_k]) = f(\mathbf{t}[x_k], \overline{\mathbf{t}[x_k]}) = f(D(x_k))$ and thus it is true for $j = k$.

It follows that $\mathbf{t^{(s)}}$ is a tuple or R that coincides with $\mathbf{t}$ on S_2, and $\mathbf{t^{(s)}}[x] = D(f(x))$ whenever $x \in S_1$. Therefore, assigning each $x \in S_1$ to $D(f(x))$ is always compatible with any assignment to S_2; since this is true for each constraint, it is true for $\mathcal{P}$ as well.

We denote by $\mathcal{IP}^{2c}(\Gamma)$ the CSP instance obtained from $\mathcal{IP}^2(\Gamma)$ by adding the unary constraints enforcing conservativity. We can interpret $\mathcal{IP}^{2c}(\Gamma)$ as the meta-problem associated with an unconstrained conservative binary operation symbol f and reuse the definitions and lemmas about closed sets of variables seen in the last section. In the hierarchy of closed sets given by Lemma 2 applied to $\mathcal{IP}^{2c}(\Gamma)$, X_{i+1} contains the variables of X_i plus two variables $x_{f(a,b)}, x_{f(b,a)}$ for some $B_{i+1} = \{a, b\} \subseteq \mathcal{D}$.

Proposition 5. *Suppose that we know a solution f_i to $\mathcal{IP}^{2c}(\Gamma)_{|X_i}$ that is maximally commutative if c-CSP(Γ) is tractable. A solution f_{i+1} to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ with the same properties can be found in polynomial time.*

Proof. The strategy is similar to the proof of Proposition 3. The two differences are that we do not have a semiuniform algorithm in general, which can be handled by Lemma 3, and the fact that we are not interested in *any* solution but in one that is maximally commutative.

Observe that if c-CSP(Γ) is tractable and $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ is 1-minimal, then its language is conservatively tractable as well and the order-2 conservative indicator problem of $\mathcal{L}(\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}})$ is $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ itself plus unconstrained variables (because X_{i+1} is closed). Therefore, by the Three Operations Theorem, a maximally commutative solution to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ is commutative on some $\{u, v\}$ if and only if there is a solution to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ that is also commutative on $\{u, v\}$. It follows from this same argument applied to X_i instead of X_{i+1} that if f_i is *not* commutative on some $(u, v) \in \mathcal{D}^2$ then either c-CSP(Γ) is NP-complete or Γ has a ternary conservative polymorphism $p_{u,v}$ that is either a majority or a minority operation on $\{u, v\}$.

Let $X_{i+1} = X_i \cup \{x_{f(a,b)}, x_{f(b,a)}\}$. We have only three assignments to examine for $(x_{f(a,b)}, x_{f(b,a)})$: (a,a), (b,b) and (a,b). The fourth assignment (b,a) is the projection onto the second argument, which does not need to be tried since we are only interested in the maximally commutative solutions to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$. For each of these assignments, we build the CSP instances $\mathcal{P}^1, \mathcal{P}^2, \mathcal{P}^3$ by adding the constraints corresponding to the possible assignments to $(x_{f(a,b)}, x_{f(b,a)})$ to $\mathcal{IP}^{2c}(\Gamma)$ and enforcing 1-minimality.

For every $j \in \{1, 2, 3\}$ and every pair $\{u, v\}$ of elements in the domain of $\mathcal{P}^j_{|X_{i+1}}$ we create an instance $\mathcal{P}^j_{uv}$ by adding the constraint $x_{f(u,v)} = x_{f(v,u)}$ to $\mathcal{P}^j$ and enforcing 1-minimality. Since the variables in $X_{i+1} \backslash X_i$ are ground in $\mathcal{P}^j_{uv}$, X_i is closed and X_i contains all singleton variables, we have $X_{i+1} \lhd X_i$ in $\mathcal{P}^j_{uv}$. By Proposition 2, f_i is a polymorphism of $\mathcal{L}(\mathcal{P}^j_{uv|X_{i+1}})$. Now, if a variable x in $\mathcal{P}^j_{uv|X_{i+1}}$ has domain size 2 and f_i is commutative on $D(x)$, by Lemma 3 we can assign x to $f_i(D(x))$ without losing the satisfiability of the instance. Once this is done, we can enforce 1-minimality again; the polymorphisms $p_{u',v'}$ guarantee that if c-CSP(Γ) is tractable, the remaining instance has a conservative generalized majority-minority polymorphism and hence a conservative 3-edge polymorphism. Using Corollary 2, we can decide if the language of $\mathcal{P}^j_{uv|X_{i+1}}$ has a conservative 3-edge polymorphism. If it does not then c-CSP(Γ) is NP-complete, and otherwise we can decide if a solution exists in polynomial time.

At this point, for every pair (u, v) of elements in the domain of some variable in $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ we know if a solution to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ that is commutative on (u, v) exists, except if $(u, v) = (a, b)$. This problem can be fixed by checking if any of $\mathcal{P}^k_{|X_{i+1}}$ or $\mathcal{P}^n_{|X_{i+1}}$ has a solution, where $\mathcal{P}^k$ and $\mathcal{P}^n$ are the subproblems corresponding to the assignments $(x_{f(a,b)}, x_{f(b,a)}) \leftarrow (a,a)$ and $(x_{f(a,b)}, x_{f(b,a)}) \leftarrow (b,b)$.

We then add the equality constraint $x_{f(u,v)} = x_{f(v,u)}$ to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ for *every* pair (u, v) (including (a, b) if applicable) such that a solution to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ that is commutative on (u, v) exists. On all other pairs, we know that f_{i+1} must project on the first argument, so we can ground the corresponding variables. If c-CSP(Γ) is tractable, then this new CSP instance $\mathcal{P}$ has a solution and it must be maximally commutative. We can solve $\mathcal{P}$ by branching on the possible assignments to $(x_{f(a,b)}, x_{f(b,a)})$ and the usual arguments using f_i, Proposition 2 and Lemma 3.

Theorem 4. *There exists a polynomial-time algorithm* A *that, given in input a constraint language Γ, decides if c-CSP(Γ) is in P or NP-complete. If c-CSP(Γ) is in P, then* A *also returns the coloured graph of Γ.*

Proof. We use Proposition 5 to find in polynomial time a conservative polymorphism f^* of Γ that is maximally commutative if c-CSP(Γ) is tractable. If the algorithm fails, then we know that c-CSP(Γ) is not tractable and the algorithm stops. Otherwise, we label every pair $\{a, b\}$ of domain elements with the colour red if f^* is commutative on $\{a, b\}$, and otherwise we use Proposition 4 to check if there is a conservative ternary polymorphism that is either majority or minority

on $\{a, b\}$. If a majority polymorphism is found then we label $\{a, b\}$ with yellow, else if a minority polymorphism is found then $\{a, b\}$ is blue, and otherwise we know that c-CSP(Γ) is NP-complete. The orientation of the red edges can be easily computed from $\mathcal{IP}^{2c}(\Gamma)$ using Lemma 3 and f^*.

5   Conclusion

We have shown that the dichotomy criterion for conservative CSP can be decided in true polynomial time, without any assumption on the arity or the domain size of the input constraint language. This solves an important question on the complexity of c-CSP among the few that remain. On the way, we have also proved that classes of conservative constraint languages defined by linear strong Mal'tsev conditions admitting a semiuniform algorithm always have a tractable meta-problem. This result is a major step towards a complete classification of meta-problems in conservative languages and complements nicely the results of [8].

It is known that Proposition 1 does not hold in general if the linearity requirement on the Mal'tsev condition is dropped, as semilattices are NP-hard to detect even in conservative constraint languages despite having a uniform algorithm [11]. The same happens if the idempotency of the Mal'tsev condition is dropped instead [8]. However, the mystery remains if the requirement for a uniform algorithm is loosened since no tractable idempotent strong linear Mal'tsev condition is known to have a hard meta-problem. This prompts us to ask if our result on conservative constraint languages can extend to the general case.

Question 1. Does there exist an idempotent strong linear Mal'tsev condition $\mathcal{M}$ that has a semiuniform polynomial-time algorithm but whose meta-problem is not in P, assuming some likely complexity theoretic conjecture?

A negative answer would imply a uniform algorithm for constraint languages with a Mal'tsev polymorphism, whose potential existence was discussed in [7]. Finally we believe that our algorithm, by producing the coloured graph in polynomial time, would be very helpful in the design of a uniform algorithm that solves every tractable conservative constraint language (should one exist).

Question 2. Does there exist a uniform polynomial-time algorithm for the class of all tractable conservative constraint languages?

References

1. Barto, L.: The dichotomy for conservative constraint satisfaction problems revisited. In: LICS, pp. 301–310. IEEE Computer Society (2011)
2. Barto, L.: The collapse of the bounded width hierarchy. J. Logic Comput. **26**, 923–943 (2015)
3. Berman, J., Idziak, P., Marković, P., McKenzie, R., Valeriote, M., Willard, R.: Varieties with few subalgebras of powers. Trans. Am. Math. Soc. **362**(3), 1445–1473 (2010)

4. Bessière, C., Carbonnel, C., Hébrard, E., Katsirelos, G., Walsh, T.: Detecting and exploiting subproblem tractability. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 468–474. AAAI Press (2013)
5. Bulatov, A.A.: Complexity of conservative constraint satisfaction problems. ACM Trans. Comput. Log. **12**(4), 24 (2011)
6. Bulatov, A.A.: Conservative constraint satisfaction re-revisited. preprint. arXiv:1408.3690v1 (2014)
7. Carbonnel, C.: The meta-problem for conservative Mal'tsev constraints. In: Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016), Phoenix, Arizona, United States, February 2016
8. Chen, H., Larose, B.: Asking the metaquestions in constraint tractability. arXiv preprint. arXiv:1604.00932 (2016)
9. Dalmau, V.: Generalized majority-minority operations are tractable. Logical Methods Comput. Sci. **2**(4), 1–15 (2006)
10. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. SIAM J. Comput. **28**(1), 57–104 (1998)
11. Green, M.J., Cohen, D.A.: Domain permutation reduction for constraint satisfaction problems. Artif. Intell. **172**(8–9), 1094–1118 (2008)
12. Idziak, P.M., Markovic, P., McKenzie, R., Valeriote, M., Willard, R.: Tractability and learnability arising from algebras with few subpowers. In: LICS, pp. 213–224. IEEE Computer Society (2007)
13. Jeavons, P., Cohen, D.A., Gyssens, M.: Closure properties of constraints. J. ACM **44**(4), 527–548 (1997)
14. Jeavons, P.G.: On the algebraic structure of combinatorial problems. Theoret. Comput. Sci. **200**, 185–204 (1998)
15. Ladner, R.E.: On the structure of polynomial time reducibility. J. ACM **22**(1), 155–171 (1975)
16. Mackworth, A.K.: Consistency in networks of relations. Artif. Intell. **8**, 99–118 (1977)
17. Post, E.L.: The Two-Valued Iterative Systems of Mathematical Logic. Annals Mathematical Studies, vol. 5. Princeton University Press, Princeton (1941)
18. Siggers, M.H.: A strong malcev condition for locally finite varieties omitting the unary type. Algebra Univers. **64**(1–2), 15–20 (2010)
19. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. In: Gottlob, G., Walsh, T. (eds.) IJCAI, pp. 1173–1178. Morgan Kaufmann, San Francisco (2003)

Propagation via Kernelization:
The Vertex Cover Constraint

Clément Carbonnel[(✉)] and Emmanuel Hebrard

LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
carbonnel@laas.fr

Abstract. The technique of kernelization consists in extracting, from an instance of a problem, an essentially equivalent instance whose size is bounded in a parameter k. Besides being the basis for efficient parameterized algorithms, this method also provides a wealth of information to reason about in the context of constraint programming. We study the use of kernelization for designing propagators through the example of the Vertex Cover constraint. Since the classic kernelization rules often correspond to dominance rather than consistency, we introduce the notion of "loss-less" kernel. While our preliminary experimental results show the potential of the approach, they also show some of its limits. In particular, this method is more effective for vertex covers of large and sparse graphs, as they tend to have, relatively, smaller kernels.

1 Introduction

The fact that there is virtually no restriction on the algorithms used to reason about each constraint was critical to the success of constraint programming. For instance, efficient algorithms from matching and flow theory [2,14] were adapted as *propagation* algorithms [16,18] and subsequently lead to a number of successful applications. NP-hard constraints, however, are often simply decomposed. Doing so may significantly hinder the reasoning made possible by the knowledge on the structure of the problem. For instance, finding a support for the NVALUE constraint is NP-hard, yet enforcing some incomplete propagation rules for this constraint has been shown to be an effective approach [5,10], compared to decomposing it, or enforcing bound consistency [3].

The concept of parameterized complexity is very promising in the context of propagating NP-hard constraints. A study of the parameterized complexity of global constraints [4], and of their pertinent parameters, showed that they were a fertile ground for this technique. For instance, a kernelization of the NVALUE constraint was introduced in [12], yielding an FPT consistency algorithm. A kernel is an equivalent instance of a problem whose size is bounded in a parameter k. If a problem has a polynomial-time computable kernel, then it is FPT since brute-force search on the kernel can be done in time $O^*(f(k))$ for some computable function f. Moreover, kernelization techniques can provide useful information about suboptimal and/or compulsory choices, which can be used to propagate. In this paper we consider the example of the *vertex cover* problem,

where we want to find a set of at most k vertices S of a graph $G = (V, E)$ such that every edge of G is incident to at least one vertex in S. This problem is a long-time favourite of the parameterized complexity community and a number of different kernelization rules have been proposed, along with very efficient FPT algorithms (the most recent being the $O(1.2738^k + k|V|)$ algorithm by Chen, Kanj and Xia [7]).

Since the complement of a minimum vertex cover is a maximum independent set, a VERTEXCOVER constraint can also be used to model variants of the maximum independent set and maximum clique problems with side constraints modulo straightforward modeling tweaks. Among these three equivalent problems, vertex cover offers the greatest variety of pruning techniques and is therefore the most natural choice for the definition of a global constraint. Through this example, we highlight the "triple" value of kernelization in the context of constraint programming:

First, some kernelization rules are, or can be generalized to, filtering rules. Since the strongest kernelization techniques rely on dominance they cannot be used directly for filtering. Therefore, we introduce the notion of *loss-less* kernelization which preserves all solutions and can thus be used in the context of constraint propagation. Moreover, we show that we can use a more powerful form of kernel, the so-called *rigid crowns* to effectively filter the constraint when the lower bound on the size of the vertex cover is tight. We discuss the various kernelization techniques for this problem in Sect. 3.

Second, even when it cannot be used to filter the domain, a kernel can be sufficiently small to speed up lower bound computation, or to find a "witness solution" and sometimes an exact lower bound. We also show that such a support can be used to obtain stronger filtering. We introduce a propagation algorithm based on these observations in Sect. 4. Along this line, the kernel could also be used to guide search, either using the witness solution or the dominance relations on variable assignments.

Third, because a kernel guarantees a size at most $f(k)$ for a parameter k, one can efficiently estimate the likelihood that these rules will indeed reduce the instance. We report experimental results on a variant of the vertex cover problem in Sect. 5. These experiments show that, as expected, kernelization techniques perform better when the parameter is small. However, we observe that the overhead is manageable, even in unfavorable cases. Moreover, one could dynamically choose whether costly methods should be applied by comparing the value of the parameter k (in our case, the upper bound of the variable standing for the size of the cover) to the input size.

2   Background and Notations

An *undirected graph* is an ordered pair $G = (V, E)$ where V is a set of vertices and E is a set of edges, that is, pairs in V. We denote the *neighborhood* $N(v) = \{u \mid \{v, u\} \in E\}$ of a vertex v, its *closed neighborhood* $N^+(v) = N(v) \cup \{v\}$ and $N(W) = \bigcup_{v \in W} N(v)$. The *subgraph* of $G = (V, E)$ induced by a subset of

vertices W is denoted $G[W] = (W, 2^W \cap E)$. An *independent set* is a set $I \subseteq V$ such that no pair of elements in I is in E. A *clique* is a set $C \subseteq V$ such that every pair of elements in C is in E. A *clique cover* T of a graph $G = (V, E)$ is a collection of disjoint cliques such that $\bigcup_{C \in T} C = V$. A *matching* is a subset of pairwise disjoint edges. A *vertex cover* of G is a set $S \subseteq V$ such that every edge $e \in E$ is incident to at least one vertex in S, i.e., $S \cap e \neq \emptyset$. The *minimum vertex cover problem* consists in finding a vertex cover of minimum size. Its decision version is NP-complete [11].

The standard algorithm for solving this problem is a simple branch and bound procedure. There are several bounds that one can use, in this paper we consider the minimum clique cover of the graph (or, equivalently, a coloring of its complement). Given a clique cover T of a graph $G = (V, E)$, we know that all but one vertices in each clique of T must be in any vertex cover of G. Therefore, $|V| - |T|$ is a lower bound of the size of the minimum vertex cover of G. The algorithm branches by adding a vertex to the cover (left branch) or adding its neighborhood to the cover (right branch).

A *constraint* is a predicate over one or several variables. In this paper we consider the vertex cover problem as a constraint over two variables: an *integer variable* K to represent the bound on the size of the vertex cover, and a *set variable* S to represent the cover itself. The former takes integer values in a *domain* $\mathcal{D}(K)$ which minimum and maximum values are denoted $\underline{K}$ and $\bar{K}$, respectively. The latter takes its values in the sets that are supersets of a *lower bound* $\underline{S}$ and subsets of an *upper bound* $\bar{S}$. Moreover, the domain of a set variable is also often constrained by its cardinality given by an integer variable $|S|$. We consider a constraint on these two variables and whose predicate is the vertex cover problem on the graph $G = (V, E)$ given as a parameter:

Definition 1 (VERTEXCOVER constraint). VERTEXCOVER$[G](K, S) \iff |S| \leq K$ & $\forall \{v, u\} \in E, \ v \in S \lor u \in S$

A *bound support* for this constraint is a solution of the VERTEXCOVER problem. Since enforcing *bound consistency* would entail proving the existence of two bound supports for each element in $\bar{S} \setminus \underline{S}$ and one for the lower bound of K, there is no polynomial algorithm unless P=NP. In this paper we consider pruning rules that are not complete with respect to usual notions of consistencies.

3   Kernelization as a Propagation Technique

3.1   Standard Kernelization

A problem is *parameterized* if each instance x is paired with a nonnegative integer k, and a parameterized problem is *fixed-parameter tractable* (FPT) if it can be solved in time $O(|x|^{O(1)} f(k))$ for some function f. A *kernelization algorithm* takes as input a parameterized instance (x, k) and creates in polynomial time a parameterized instance (x', k') of the same problem, called the *kernel*, such that

(i) (x', k') is satisfiable if and only if (x, k) is satisfiable;
(ii) $|x'| \leq g(k)$ for some computable function g, and
(iii) $k' \leq h(k)$ for some computable function h.

While this formal definition does not guarantee that the kernel is a subinstance of (x, k), in graph theory kernelization algorithms often operate by applying a succession of dominance rules to eliminate vertices or edges from the graph. In the case of vertex cover, the simplest dominance rule is the *Buss rule*: if a vertex v has at least $k + 1$ neighbors, then v belongs to every vertex cover of size at most k; we can therefore remove v from the graph and reduce k by one. Applying this rule until a fixed point yields an elementary kernel that contains at most k^2 edges and $2k^2$ non isolated vertices [6]. A more refined kernelization algorithm works using structures called crowns. A *crown* of a graph $G = (V, E)$ is a partition (H, W, I) of V such that

(i) I is an independent set;
(ii) There is no edge between I and H, and
(iii) There is a matching M between W and I of size $|W|$.

Every vertex cover of $G[W \cup I]$ has to be of size at least $|W|$ because of the matching M. Since I is an independent set, taking the vertices of W over those of I into the vertex cover is always a sound choice: they would cover all the edges between W and I at minimum cost and as many edges in $G[H \cup W]$ as possible. A simple polynomial-time algorithm that finds a crown greedily from a maximal matching already leaves an instance $G[H]$ with at most $3k$ vertices [1]. A stronger method using linear programming yields a (presumably optimal) kernel of size $2k$ [15].

3.2   Loss-Less Kernelization

The strongest kernelization rules correspond to dominance relations rather than inconsistencies. However, the Buss rule actually detects inconsistencies, that is, vertices that must be in the cover. We call this type of rules *loss-less* as they do not remove solutions. We can extend this line of reasoning by considering rules that do not remove solutions close to the optimum: for the VERTEXCOVER constraint, the variable K is likely to be minimized and the situation where all solutions are close to the optimum will inevitably arise. This can be formalized in the context of *subset minimization problems*, which ask for a subset S with some property π of a given universe U such that $|S| \leq k$. In the next definition we denote by `opt` the cardinality of a minimum-size solution.

Definition 2. *Given an integer z and a subset minimization problem parameterized by solution size k, a z-loss-less kernel is a partition (H, F, R, I) of the universe U where*

– *F is a set of* forced *items, included in every solution of size at most `opt`+z;*
– *R is a set of* restricted *items, intersecting with no solution of size at most `opt`+z;*

– H is a residual problem, whose size is bounded by a function in k and
– I is a set of indifferent elements, i.e., if $i \in I$, then ϕ is a solution of size at most $k-1$ if and only if $\phi \cup i$ is a solution.

An ∞-loss-less kernel is simply said to be *loss-less*. The Buss kernel is a loss-less kernel for vertex cover that never puts any vertices in R (F contains vertices of degree strictly greater than k, and I contains isolated vertices). In the case of vertex cover, the set R is always empty unless $z = 0$. Note that *loss-free* kernels introduced in the context of backdoors [17] are different since they only preserve *minimal* solutions; for subset minimization problems those kernels are called *full kernels* [9].

A kernel for vertex cover that preserves all minimum-size solutions has been introduced in [8]. In our terminology, this corresponds to a 0-loss-less kernel. Interestingly, this kernelization is based on a special type of crown reduction but yields a kernel of size $2k$ (matching the best known bound for standard kernelization). The idea is to consider only crowns (H, W, I) such that W is the *only* minimum-size vertex cover of $G[W \cup I]$, as for this kind of crown vertices of W are always a *strictly* better choice that those of I. Those crowns are said *rigid*. The authors present a polynomial-time algorithm that finds the (unique) rigid crown (H, W, I) such that H is rigid crown free and has size at most $2k$. Their algorithm works as follows. First, build from $G = (V, E)$ the graph B_G with two vertices v_l, v_r for every $v \in V$ and two edges $\{v_l, u_r\}, \{u_l, v_r\}$ for every edge $\{v, u\} \in E$. Compute a maximum matching M of B_G (which can be done in polynomial time via the Hopcroft-Karp algorithm [14]). Then, if D is the set of all vertices in B_G that are reachable from unmatched vertices via M-alternating paths of even length, a vertex v in G belongs to the independent set I of the rigid crown if and only if v_l and v_r belong to D. This algorithm is well suited to constraint propagation as bipartite matching algorithms based on augmenting paths are efficient and incremental.

3.3   Witness Pruning

Last, even if the standard kernel uses dominance relations, it can indirectly be used for pruning. By reducing the size of the problem it often makes it possible to find an optimal vertex cover relatively efficiently. This vertex cover gives a valid (and maximal) lower bound. Moreover, given an optimal cover S we can find inconsistent values by asserting that some vertices must be in any cover of a given size.

Theorem 1. *If S is an optimal vertex cover of $G = (V, E)$ such that there exists $v \in S, J \subseteq N(v) \setminus S$ with $N(J) \subseteq N^+(v)$ then any vertex cover of G either contains v or at least $|S| + |J| - 1$ vertices.*

Proof. Let k be an upper bound on the size of the vertex cover, $v \in S$ be a vertex in an optimal vertex cover S. Consider $J \subseteq N(v) \setminus S$ such that $N(J) \subseteq N^+(v)$. Suppose there exists a vertex cover S' such that $|S'| < |S| + |J| - 1$ and $v \notin S'$. S' must contain every node in $N(v)$ and hence in J. However, we can build a

vertex cover of size at most $|S| - 1$ by replacing J by v, since $V \setminus S$ and thus J are independent sets.   □

If we can manage to find a minimum vertex cover S, for instance when the kernel is small enough so that it can be explored exhaustively, Theorem 1 entails a pruning rule. If we find a vertex $v \in S$ and a set $J \in N(v) \setminus S$ with $N(J) \subseteq N^+(v)$ and $|J| > k - |S|$ then we know that v must be in all vertex covers of size $\leq k$.

4   A Propagation Algorithm for VERTEXCOVER

In this section we give the skeleton of a propagation algorithm for the VERTEXCOVER constraint based on the techniques discussed in Sect. 3.

Algorithm 1. PropagateVertexCover$(S, K, G = (V, E), \lambda, \omega)$

---

1 $\underline{S} \leftarrow \underline{S} \cup N(V \setminus \overline{S})$;
2 $H^r, F^r \leftarrow$ BussKernel$(G[\overline{S} \setminus \underline{S}])$;
3 **if** $\omega \not\subseteq \overline{S} \ \vee \ |\omega \cup \underline{S}| \geq \overline{K}$ **then**
4 | $H^k, W^k \leftarrow$ Kernel(H^r);
5 | **if** $\lambda > 0$ **then** $\omega \leftarrow F^r \cup W^k \cup$ VertexCover(H^k, λ);
6 | **if** ω *is optimal* **then** $\underline{K} \leftarrow |\omega|$;
7 | **else** $\underline{K} \leftarrow \max(\underline{K}, |F^r| + |F^k| +$ LowerBound$(H^k))$;

8 **if** $\underline{K} = \overline{K}$ **then**
9 | $H^r, F^r, R^r \leftarrow$ RigidKernel$(G[\overline{S} \setminus \underline{S}])$;
10 | $\overline{S} \leftarrow \overline{S} \setminus R^r$;

11 **else if** ω *is optimal* & $\overline{K} - \underline{K} \leq 2$ **then** $\underline{S}, \overline{S} \leftarrow$ WitnessPruning(G, ω) ;
12 $\underline{S} \leftarrow \underline{S} \cup F^r$;

Algorithm 1 takes as input the set variable S standing for the vertex cover, an integer variable K standing for the cardinality of the vertex cover, and three parameters: the graph $G = (V, E)$, an integer λ, and a "witness" vertex cover ω initialised to V.

The pruning in Line 1 is a straightforward application of the definition: the neighborhood of vertices not in the cover must be in the cover. Then, in Line 2, we apply the ∞-loss-less kernelization (Buss rule) described in Sect. 3.2 yielding a pair with a residual graph H^r and a set of nodes F^r that must be in the cover.

Next, if Condition 3 fails, there exists a vertex cover ($\omega \cup \underline{S}$) of size strictly less than $\overline{K}$. As a result, the pruning from rigid crowns cannot apply. When the cover witness is not valid, we compute, in Line 4, a standard kernel with the procedure Kernel(G) using crowns, as explained in Sect. 3.1. We then use this kernel to compute, in Line 5, a new witness using the procedure VertexCover(G, λ) which is the standard brute-force algorithm described in Sect. 2. We stop the procedure

when we find a vertex cover whose size is strictly smaller than the current upper bound, or when the search limit of λ, in number of nodes explored by the branch &bound procedure, is reached. In the first case, we know that the lower bound cannot be tight hence the constraint cannot fail nor prune further than the lossless kernel. The second stopping condition is simply used to control the amount of time spent within the brute-force procedure.

If the call to the brute-force procedure was complete, we can conclude that the witness cover is optimal and therefore a valid lower bound (Line 6). Otherwise, we simply use the lower bound computed at the root node by `VertexCover`, denoted `LowerBound` in Line 7. If the lower bound is tight, then we can apply the pruning from rigid crowns as described in Sect. 3.2. Algorithm `RigidKernel` returns a triple H^r, F^r, R^r of residual, forced and restricted vertices, respectively. Finally we apply a restriction to pairs of the pruning corresponding to Theorem 1 in Line 11, and apply the pruning on the lower bound of S corresponding to the forced nodes computed by `BussKernel` and/or `RigidKernel`.

5   Experimental Evaluation

We experimentally evaluated our propagation algorithm on the "balanced vertex cover problem". We want to find a minimum vertex cover which is balanced according to a partition of the vertices. For instance, the vertex cover may represent a set of machines to shut down in a network so that all communications are interrupted. In this case, one might want to avoid shutting down too many machines of the same type, or same client, or in charge of the same service, etc. By varying the degree of balance we can control the similarity of the problem to pure minimum vertex cover. We used a range of graphs from the `dimacs` and `snap` repositories. For each graph $G = (V, E)$, we post a VERTEXCOVER constraint on the set variable $\emptyset \subseteq S \subseteq V$.

Then, we compute (uniformly at random) a balanced 4-partition $\{s_1, s_2, s_3, s_4\}$ of the vertices and we post the following constraints: $\max(\{|s_i \cap S| \mid 1 \leq i \leq 4\}) - \min(\{|s_i \cap S| \mid 1 \leq i \leq 4\}) \leq b$. For each graph instance, we generated 3 instances for $b \in \{0, 4, 8\}$ denoted "tight", "medium" and "loose" respectively. However, the classes `p2p` and `ca-` are much too large for these values to make sense. In this case we used three ratios $0.007, 0.008$ and 0.009 of the number of nodes instead.

We compared 5 methods, all implemented in Mistral [13] and ran on CORE I7 processors with a time limit of 5 min:

Decomposition is a simple decomposition in 2-clauses and a cardinality constraint. **Clique Cover** uses only Buss kernelization and the clique cover lower bound. It corresponds to non-colored lines in Algorithm 1. The witness is initialised to V and never changes, and Line 4 is replaced by a simple identity $H^k \leftarrow H^r$. **Kernel Pruning** uses kernelization, but no witness cover. It corresponds to Algorithm 1 minus the instruction line 11, with λ set to 0. **Kernel & witness** uses kernelization, and the witness cover for the lower bound $\underline{K}$. It corresponds to Algorithm 1 minus the instruction line 11, with λ set to 5000. VERTEXCOVER is Algorithm 1 with λ set to 5000.

Table 1. Comparison of approaches on the "Balanced Vertex Cover" problem.

	Decomposition				Clique Cover				Kernel Pruning				Kernel & witness				VERTEXCOVER			
	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd
balancing constraint: tight																				
3 kel	2	2.00	9.7	0.4M	2	2.00	10.6	0.2M	2	2.00	9.1	0.2M	2	2.00	26.6	0.1M	2	2.00	41.0	0.1M
15 p_h	12	5.73	8.6	0.5M	10	5.20	15.6	1.1M	11	5.20	11.2	0.6M	11	4.67	27.7	0.4M	11	4.67	28.8	0.4M
12 bro	9	3.67	0.1	11K	9	3.67	0.1	4K	9	3.67	0.1	3K	9	3.67	0.1	2K	9	3.67	0.2	2K
4 joh	1	0.00	0.1	10K	1	0.00	0.0	1K	1	0.00	0.0	1K	1	0.00	0.0	971	1	0.00	0.0	937
15 san	15	10.87	12.2	1.8M	11	9.80	13.3	1.9M	11	9.80	13.7	1.1M	11	9.80	10.8	0.6M	11	9.80	12.4	0.6M
7 c-f	3	10.29	0.2	9K	3	10.29	0.2	18K	3	10.29	0.1	7K	3	10.29	0.1	7K	3	10.29	0.1	7K
6 ham	4	9.00	26.3	2.2M	3	9.00	3.1	0.3M	3	9.00	3.4	0.2M	3	9.00	5.1	0.2M	3	9.00	5.1	0.2M
32 gra	29	40.47	24.6	2.5M	28	40.47	19.8	3.5M	28	39.22	17.6	2.0M	28	40.47	18.9	1.5M	28	41.22	9.8	0.5M
4 man	3	91.00	1.1	33K	3	91.00	1.1	51K	3	91.00	0.9	31K	3	91.00	1.4	31K	3	91.00	1.5	30K
5 mul	5	8.40	7.2	1.8M	4	7.60	41.4	6.3M	3	7.60	24.6	2.1M	3	7.60	25.1	2.1M	3	7.60	19.2	1.7M
3 fps	3	105.00	0.1	7K	3	103.67	40.5	4.2M	3	103.67	56.0	3.2M	3	103.67	61.0	3.2M	3	103.67	14.9	0.8M
3 zer	3	44.67	11.9	2.6M	3	44.67	11.4	1.6M	3	44.67	14.7	1.4M	3	44.67	13.9	1.4M	3	44.67	8.2	0.9M
3 ini	3	191.33	57.5	6.1M	3	191.33	72.7	6.1M	3	191.33	82.3	6.1M	3	191.33	82.6	6.1M	3	191.33	82.6	6.1M
5 p2p	5	38.60	1.0	8K	5	22.80	36.1	23K	2	11.80	2.8	11K	2	11.80	3.1	11K	2	11.80	3.4	11K
5 ca-	5	14.40	31.6	0.2M	4	9.00	35.6	0.2M	4	1.80	99.3	0.2M	3	2.60	102.3	0.2M	3	1.80	96.1	0.2M
balancing constraint: medium																				
3 kel	2	1.67	24.1	1.2M	2	0.67	35.9	1.0M	2	0.67	54.7	1.0M	2	0.00	32.8	2K	2	0.00	32.1	2K
15 p_h	12	3.07	21.5	1.2M	10	1.27	24.3	0.7M	11	1.27	34.4	0.6M	10	0.87	18.6	60K	10	0.87	18.8	59K
12 bro	9	0.83	15.6	1.9M	8	0.17	17.8	1.0M	8	0.17	25.4	1.0M	8	0.17	23.9	451	8	0.17	22.2	450
4 joh	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11
15 san	15	8.33	35.6	5.0M	7	2.67	30.4	2.4M	7	2.73	33.5	1.6M	7	1.53	42.8	0.4M	7	1.53	43.1	0.4M
7 c-f	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40
6 ham	4	4.67	0.2	53K	2	4.67	0.0	1K	2	4.67	0.0	360	2	4.67	0.0	359	2	4.67	0.0	359
32 gra	26	29.28	32.8	2.7M	22	26.50	21.9	2.2M	22	24.44	23.8	1.4M	22	24.25	21.5	0.9M	22	24.25	23.0	0.9M
4 man	3	89.00	29.3	1.3M	3	88.75	44.6	1.6M	3	88.75	21.1	0.6M	2	88.50	29.6	0.6M	2	88.50	33.4	0.6M
5 mul	5	1.20	0.3	61K	1	1.20	0.0	682	1	1.20	0.0	682	1	1.20	0.0	682	1	1.20	0.0	560
3 fps	3	103.00	0.0	250	1	102.67	0.0	429	1	102.67	0.0	404	1	102.67	0.0	404	1	102.67	0.0	261
3 zer	3	3.33	37.4	8.2M	1	3.00	11.6	1.5M	1	3.00	25.4	1.5M	1	3.00	14.5	1.5M	1	3.00	14.5	1.5M
3 ini	3	189.00	0.6	65K	1	189.00	0.0	4K	1	189.00	0.0	3K	1	189.00	0.0	3K	1	189.00	0.0	3K
5 p2p	5	35.40	1.0	8K	5	15.80	38.3	26K	1	4.40	3.3	11K	1	4.40	3.5	11K	1	4.40	3.8	11K
5 ca-	5	14.40	0.7	5K	4	8.60	2.3	8K	3	0.40	72.6	18K	2	1.20	74.1	16K	2	0.40	64.0	15K
balancing constraint: loose																				
3 kel	2	1.67	43.3	1.8M	2	0.67	20.1	0.6M	2	0.67	30.4	0.6M	2	0.00	27.7	447	2	0.00	28.0	419
15 p_h	12	2.40	20.6	1.2M	10	0.73	32.1	1.0M	11	0.73	47.1	1.0M	9	0.27	18.0	3K	9	0.27	18.0	3K
12 bro	9	0.67	16.2	1.9M	8	0.00	10.6	0.7M	8	0.00	15.8	0.7M	8	0.00	13.6	264	8	0.00	13.6	264
4 joh	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11
15 san	15	8.20	28.1	4.0M	7	2.13	40.9	2.3M	7	2.27	29.6	1.4M	5	0.27	36.8	3K	5	0.27	36.8	3K
7 c-f	2	0.71	0.0	1K	0	0.00	0.6	98K	0	0.00	0.1	7K	0	0.00	0.2	7K	0	0.00	0.2	7K
6 ham	4	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118
32 gra	23	18.97	29.4	1.9M	17	12.84	12.9	0.8M	17	12.06	15.6	0.5M	17	11.56	14.5	66K	17	11.50	17.3	0.1M
4 man	3	88.50	38.1	0.8M	3	88.50	8.5	0.3M	3	87.75	30.6	0.8M	2	87.00	43.5	0.8M	2	86.50	52.4	0.8M
5 mul	5	0.00	0.0	93	0	0.00	0.0	92	0	0.00	0.0	91	0	0.00	0.0	91	0	0.00	0.0	91
3 fps	3	1.67	1.1	0.1M	1	1.00	12.6	1.0M	1	1.00	13.4	0.5M	1	1.00	13.9	0.5M	1	0.67	53.5	2.1M
3 zer	3	2.00	0.2	48K	0	1.00	2.6	0.4M	0	1.00	2.7	0.2M	0	1.00	2.6	0.2M	0	1.00	0.6	58K
3 ini	3	0.67	6.9	0.5M	0	0.00	20.9	1.3M	0	0.00	28.9	1.0M	0	0.00	31.9	1.0M	0	0.00	11.6	0.5M
5 p2p	5	33.00	1.0	8K	5	13.40	38.3	26K	1	2.00	3.4	11K	1	2.00	3.5	11K	1	2.00	3.9	11K
5 ca-	5	14.40	0.7	5K	4	8.60	2.5	8K	3	0.40	74.0	18K	2	1.20	73.9	16K	2	0.20	69.2	16K

The results of these experiments are reported in Table 1. Instances are clustered by classes whose cardinality is given in the first column. These classes are ordered from top to bottom by decreasing ratio of minimum vertex cover size over number of nodes. We report four values for each class and each method: '#s' is the number of instances of the class that were not solved to optimality, 'gap' is the average gap w.r.t. the smallest vertex cover found, 'cpu' and '#nd' are mean CPU time in seconds and number of nodes visited, respectively, until finding the best solution. Notice that CPU times and number of nodes are then

only comparable when the objective values (gaps) are equal. We color the tuples $\langle$#s, gap, cpu, #nd$\rangle$ that are lexicographically minimum for each class[1].

Instances with same value of b are grouped in the same sub-table. The "shift" of colored cells from left to right when going from top to bottom in each subtable was to be expected since the kernelization is more effective on instances with small vertex cover. It should be noted that many instances from the `dimacs` repository are extremely adverse to our method as they tend to have very large vertex covers. On the other hand, kernelization is very effective on large graphs from `snap`.

We can also observe another shift of colored cells from left to right when moving to a subtable to the next. This was also an expected outcome since the pruning on this constraint becomes more prevalent when the problem is closer to pure vertex cover.

Last, we can observe that every reasoning step (0-loss-less kernels, lower bound from the witness and pruning from the witness) improves the overall results.

6   Conclusion

We have shown that the kernelization techniques can be an effective way to reason about NP-hard constraints that are fixed parameter tractable. In order to design a propagation algorithm we introduced the notion of loss-less kernel and outlined several ways to benefit from a small kernel. Our experimental evaluation on the VERTEXCOVER constraint shows the promise of this approach.

References

1. Abu-Khzam, F.N., Fellows, M.R., Langston, M.A., Suters, W.H.: Crown structures for vertex cover kernelization. Theory Comput. Syst. **41**(3), 411–430 (2007)
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall Inc., Upper Saddle River (1993)
3. Beldiceanu, N.: Pruning for the minimum constraint family and for the number of distinct values constraint family. In: CP, pp. 211–224 (2001)
4. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Quimper, C.-G., Walsh, T.: The parameterized complexity of global constraints. In: AAAI, pp. 235–240 (2008)
5. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Filtering algorithms for the NValue constraint. Constraints **11**(4), 271–293 (2006)
6. Buss, J.F., Goldsmith, J.: Nondeterminism within p*. SIAM J. Comput. **22**(3), 560–572 (1993)
7. Chen, J., Kanj, I.A., Xia, G.: Improved parameterized upper bounds for vertex cover. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 238–249. Springer, Heidelberg (2006)
8. Chlebík, M., Chlebíková, J.: Crown reductions for the minimum weighted vertex cover problem. Discrete Appl. Math. **156**(3), 292–312 (2008)

---

[1] With a "tolerance" of 1 s and 1 % nodes.

9. Damaschke, P.: Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. Theor. Comput. Sci. **351**(3), 337–350 (2006)
10. Fages, J.-G., Lapègue, T.: Filtering AtMostNValue with difference constraints: application to the shift minimisation personnel task scheduling problem. In: CP, pp. 63–79 (2013)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York (1979)
12. Gaspers, S., Szeider, S.: Kernels for global constraints. In: IJCAI, pp. 540–545 (2011)
13. Hebrard, E.: Mistral, a constraint satisfaction library. In: The Third International CSP Solver Competition, pp. 31–40 (2008)
14. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4), 225–231 (1973)
15. Nemhauser, G.L., Trotter Jr., L.E.: Vertex packings: structural properties and algorithms. Math. Program. **8**(1), 232–248 (1975)
16. Régin, J.-C.: A filtering algorithm for constraints of difference in CSPs. In: AAAI, pp. 362–367 (1994)
17. Samer, M., Szeider, S.: Backdoor trees. In: AAAI, vol. 8, pp. 13–17 (2008)
18. van Hoeve, W.-J., Pesant, G., Rousseau, L.-M.: On global warming: flow-based soft global constraints. J. Heuristics **12**(4–5), 347–373 (2006)

Breaking Symmetries in Graphs:
The Nauty Way

Michael Codish[1]([✉]), Graeme Gange[2], Avraham Itzhakov[1],
and Peter J. Stuckey[2,3]

[1] Department of Computer Science, Ben-Gurion University of the Negev,
Beer Sheva, Israel
`mcodish@cs.bgu.ac.il`
[2] Department of Computing and Information Systems,
The University of Melbourne, Melbourne, Australia
[3] Data61 CSIRO, Melbourne, Australia

Abstract. Symmetry breaking is an essential component when solving graph search problems as it restricts the search space to that of canonical representations. There are an abundance of powerful tools, such as `nauty`, which apply to find the canonical representation of a given graph and to test for isomorphisms given a set of graphs. In contrast, for graph search problems, current symmetry breaking techniques are partial and solvers unnecessarily explore an abundance of isomorphic parts of the search space. This paper is novel in that it introduces complete symmetry breaking for graph search problems by modeling, in terms of constraints, the same ideas underlying the algorithm applied in tools like `nauty`. Whereas `nauty` *tests* given graphs, symmetry breaks restrict the search space and apply during *generation*.

1 Introduction

Many problems, particularly in combinatorics, reduce to asking whether some graph with a given property exists. Such "graph search" or "graph existence" problems are notoriously difficult, in no small part due to the extremely large number of symmetries in graphs. General approaches to graph search problems involve either explicitly enumerating all (non-isomorphic) graphs and checking each for the given property, or encoding the problem for some general-purpose discrete satisfiability solver (i.e. SAT, integer programming, constraint programming), which does the enumeration implicitly. In this paper, we are largely concerned with this second approach.

To avoid symmetries in explicit enumeration approaches, ideally one designs a procedure which generates exactly one graph in each equivalence class. The classic "orderly generation" approach, due to Read [1], imposes a lexicographic order over matrix elements and systematically constructs canonical adjacency matrices of size $n+1$ from the canonical matrices of size n. For an example, in [2], the authors address the problem: does there exist a graph with 11 vertices which has a total magic labeling (TML)? To provide the negative answer the authors

test each one of the 1,018,997,864 canonical graphs with 11 vertices and report that this task requires 13,595 days of CPU computation. Given that there are 165,091,172,592 canonical graphs with 12 vertices it clear that canonical graph enumeration based approaches cannot scale.

In contrast, symmetry breaking in SAT or CP [3–6] is done by adding additional constraints to eliminate non-canonical graphs.[1] Existing symmetry-breaking predicates are typically based on variants of the lexicographic ordering [4]. Incomplete symmetry breaks under this ordering are straightforward and compact, but leave many non-canonical graphs in each equivalence class. Complete symmetry breaks, on the other hand, are extremely large. Indeed, as deciding lexicographic canonicity of an adjacency matrix is NP-hard, the existence of a compact complete symmetry break seems unlikely.

By looking at vertex degrees and related properties, it is often possible to very quickly conclude that two graphs are not isomorphic. In fact, most non-isomorphic graphs may be distinguished in this way. It turns out that exploiting *structural* properties of graphs is critical in testing equivalence or finding canonical representations, and gave rise to a family of astonishingly effective isomorphism and canonical labeling tools. Many graph search problems instead generate candidate solutions, which are then reduced to canonical form using canonical label-ling tools such as nauty [9], bliss [10], or saucy [11]. This approach can be highly effective since these tools are amazingly efficient, but it can be overwhelmed by generating enormous numbers of copies of isomorphic graphs. For example there are 36,028,797,018,963,968 adjacency matrices on 11 vertices which is considerably more than the number of canonical graphs.

Ideally we would like to impose constraints defining the properties of the graph we are searching for *together with* a compact constraint on the structural properties of the graph to eliminate all non-canonical solutions. Then we could exploit state of the art declarative solvers, to solve graphs problems with arbitrary constraints and objective functions without being overwhelmed by symmetry in the search.

This paper makes two contributions. First, we introduce a polynomial sized SAT encoding of a partial symmetry-breaking predicate which exploits structural information in the style of nauty which eliminates many more non-canonical graphs than standard lex-based approaches. When combined with lexicographic symmetry breaking this predicate remains polynomial and breaks even more symmetries. Second, we illustrate how a technique, first presented in [12], can be generalized to compute complete symmetry-breaking predicates which enhance the nauty style structural break. While these predicates could be exponential in size, we show that they are very small in practice. We present experimental results to demonstrate the impact of both types of nauty style symmetry breaks: partial and complete.

The computations throughout the paper are performed using the finite-domain constraint compiler BEE [13] which compiles constraints to CNF, and

---

[1] We restrict our consideration here to *static* symmetry breaking, rather than dynamic approaches such as SBDS [7] or LDSB [8].

solves it applying an underlying SAT solver. We use Glucose 4.0 [14] and Clasp 3.1.3 [15] as the underlying SAT solvers and specify for each computation which solver was used. All computations were performed on a cluster of Intel E8400 cores, each clocked at 2 GHz, able to run a total of 790 parallel threads. Each of the cores in the cluster has computational power comparable to a core on a standard desktop computer. Each SAT instance is run on a single thread.

In Sect. 2 we present preliminaries on graphs, graph isomorphism, on the nauty approach to graph isomorphism, and on symmetry breaking in graph search problems. Section 3 describes a symmetry breaking predicate which exploits structural information, emulating the nauty algorithm. This section also presents an experimental evaluation comparing the new symmetry breaks with other existing techniques. Finally, Sect. 4 concludes.

2   Preliminaries

2.1   Graphs, Permutations, Graph Isomorphism, Canonical Graphs

Throughout this paper we consider finite and simple graphs (undirected with no self loops). The set of simple graphs on n nodes is denoted $\mathcal{G}_n$. We assume that the vertex set of a graph, $G = (V, E)$, is $V = \{1, \ldots, n\}$ and represent G by its $n \times n$ adjacency matrix A defined by $A_{i,j} = (1$ if $(i, j) \in E$ else 0$)$. We write A_i to denote the ith row of A.

The set of permutations $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ is denoted S_n. For convenience, we shall use $\pi_{i,j}$ to denote the permutation swapping i with j that maps every other element to itself.

For $G = (V, E) \in \mathcal{G}_n$ and $\pi \in S_n$, we define $\pi(G) = \{V, \{(\pi(u), \pi(v))|(u, v) \in E)\}$. Permutations act on adjacency matrices in the natural way: If A is the adjacency matrix of a graph G, then $\pi(A)$ is the adjacency matrix of $\pi(G)$ obtained by simultaneously permuting with π the rows and columns of A.

Two graphs $G_1, G_2 \in \mathcal{G}_n$ are isomorphic, denoted $G_1 \approx G_2$, if there exists a permutation $\pi \in S_n$ such that $G_1 = \pi(G_2)$. Sometimes we write $G_1 \approx_\pi G_2$ to emphasize that π is the permutation such that $G_1 = \pi(G_2)$. For sets of graphs H_1, H_2, we say that $H_1 \approx H_2$ if for every $G_1 \in H_1$ (likewise in H_2) there exists $G_2 \in H_2$ (likewise in H_1) such that $G_1 \approx G_2$. The equivalence classes of $\mathcal{G}$ modulo $\approx$ is denoted $\mathcal{G}_n^{\approx}$.

It is usual to define the canonical representation of (an equivalence class of) a graph in terms of some total ordering. A classic choice is the lexicographic ordering on graphs.

Definition 1 (lex ordering graphs). *Let $G_1, G_2 \in \mathcal{G}_n$ and let s_1, s_2 be the strings obtained by concatenating the rows of the upper triangular parts of their corresponding adjacency matrices A_1, A_2 respectively. Then, $G_1 \preceq_{lex} G_2$ if and only if $s_1 \preceq_{lex} s_2$. We also write $A_1 \preceq_{lex} A_2$.*

The canonical representation of a graph, with respect to a given total order, is then the minimal element of its equivalence class.

Definition 2 (canonicity). *The canonical representation of a graph $G \in \mathcal{G}_n$ with respect to a total ordering $\preceq$ is $can_{\preceq}(G) = \min_{\preceq} \{ \pi(G) \,|\, \pi \in S_n \}$.*

The combination of Definitions 1 and 2 provides a simple notion of canonicity defined in terms of lexical ordering of graphs which is often attributed to Read [1]. However, this definition completely ignores all of the structural information present in the graphs. A simple example of structural information is to focus on the degrees of vertices. Definitions that take advantage of structural properties of graphs simplify the processes of testing for graph isomorphism and testing for canonicity.

A *structural property* of a graph G is one which is invariant under permutation. In particular, if the property holds for a vertex v of G, then for a permutation π, it will hold also for $\pi(v)$ of $\pi(G)$. For simplicity, we will view a structural property of G as a mapping μ_G of graph vertices to integers such that for any permutation π and vertex v, $\mu_G(v) = \mu_{\pi(G)}(\pi(v))$. We often omit the subscript and write μ. For intuition, consider the structural property of vertex degree $\mu = \mathsf{deg}$ where $\mathsf{deg}_G(v)$ is the degree of v in G. For a structural property, μ_G, on a graph G with vertices $V = \{1, \ldots, n\}$, we denote $\bar{\mu}_G = \langle \mu_G(1), \ldots, \mu_G(n) \rangle$. Given μ, we introduce a total ordering $\preceq_{\mu}$ on graphs defined as follows.

Definition 3 (μ ordering graphs). *Let μ be a structural property and $G_1, G_2 \in \mathcal{G}_n$. Then, $G_1 \preceq_{\mu} G_2 \iff (\bar{\mu}_{G_1} \succ_{lex} \bar{\mu}_{G_2}) \vee ((\bar{\mu}_{G_1} = \bar{\mu}_{G_2}) \wedge (G_1 \preceq_{lex} G_2))$.*

It follows, from Definitions 2 and 3 that the canonical graph $G' = can_{\preceq_{\mu}}(G)$ has the property that $\mu_{G'}$ is sorted in decreasing order. Hence, throughout the paper, when given a structural property μ, we will focus on graphs G such that μ_G is sorted in decreasing order. The reason that we take the reverse lexicographic order on the integer vectors in Definition 3 is to be consistent with the notion of a degree sequence [16]. When $\mu = \mathsf{deg}$ and G is canonical, then μ_G is the degree sequence of G.

Definition 4 (graph partitioning). *A partitioning $\mathcal{P}$ of (vertex set) $V = \{1, \ldots, n\}$ is a sequence of k disjoint sets $\langle P_1, \ldots, P_k \rangle$ such that $V = P_1 \cup \cdots \cup P_k$. We refer to these sets as* parts *(rather than* partitions*) to avoid possible confusion. We also represent $\mathcal{P}$ as a sequence of integer values, $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ such that $1 \leq p_i \leq k$ for $1 \leq i \leq n$. Here, $p_i = j$ means that vertex i is in part j. So, for $1 \leq j \leq k$ we have $\mathcal{P}_j = \{ i \in V \,|\, p_i = j \}$. To remove ambiguity from this representation, we assume that $\mathcal{P}$ is the smallest sequence (in the reverse lexicographic order) defining the given partitioning. We shall use $\mathcal{P}(i)$ to denote the part containing vertex i (that is, p_i). When referring to a sequence of partitionings, $\mathcal{P}^k$ shall be used to denote the kth element of the sequence.*

The following example illustrates how structural information can be applied to partition the nodes of a graph. In the example, one can view the degree sequence of the graph as inducing a partitioning.
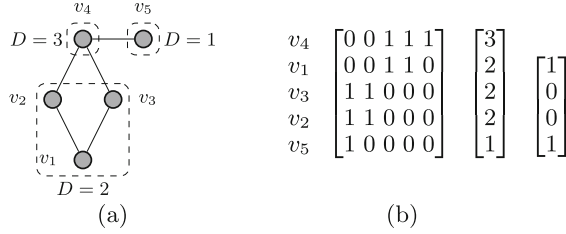
Fig. 1. (a) A graph with partitions induced by degree, and (b) its canonical adjacency matrix under $\preceq_{\mathsf{deg}}$ with its degree sequence and binary partitioning to the right.

Example 1. Consider the graph G shown in Fig. 1(a). Discriminating vertices by degree establishes the partitioning $\langle \{v_5\}, \{v_1, v_2, v_3\}, \{v_4\} \rangle$. The canonical graph $can_{\mathsf{deg}}(G)$ will contain these parts in order. Finding the canonical representation of G requires finding the permutation of $\{v_1, v_2, v_3\}$ which minimises the adjacency matrix. The resulting canonical matrix is shown in Fig. 1(b) together with the corresponding degree sequence. On the right is a binary representation of the partitioning. A one at row $i < n$ indicates that vertex $i + 1$ starts a new part, and a zero, that it is in the same part. □

2.2 The Nauty Approach

The dominant approach for constructing canonical representations of graphs is the nauty algorithm, due to McKay [9,17]. Our approach draws on the design of this algorithm. The algorithm consists of three phases applied in alternation to find a canonical representation of a given graph. Taking a very simplified view of the algorithm, we describe it in terms of two phases. The third phase, called *automorphism detection* [9,17], is not detailed in our presentation. First, nauty partitions the vertices of the graph based on structural information. Then, it searches for a canonical representation given the partitioning of the first phase.

Phase One. Structural information in nauty: For a given graph, $G = (V, E)$, The algorithm extracts structural information derived from vertex degrees to incrementally refine a partitioning of the vertices starting from a single part, $\mathcal{P} = \langle V \rangle$. We first introduce the notion of degree by partition.

Definition 5 (degree by partition). *Let* $G = (V, E)$, $V = \{1, \ldots, n\}$, $\mathcal{P} = \langle P_1, \ldots, P_k \rangle$ *be a partitioning of* V, *and* $v \in V$. *Then,* $\mathsf{deg}(v, \mathcal{P}) = \langle d_1, \ldots, d_k \rangle$ *where* $d_i = \left| \{ u \in P_i \mid (u, v) \in E \} \right|$ *counts the degrees of* v *into the parts of* $\mathcal{P}$.

Algorithm 1 is what happens in the first phase of nauty. In the terminology of [9,17], the partitioning computed by Algorithm 1 is said to be *equitable*; and a partitioning is said to be *discrete* if every equivalence class is a singleton.

Algorithm 1. nauty phase 1: partitioning

1: **procedure** PARTITION-REFINEMENT($G = (V, E)$)
2:      **init** $\mathcal{P} = \langle V \rangle$
3:      **while** $\mathcal{P} \neq$ REFINE($G,\mathcal{P}$) **do**
4:          $\mathcal{P} \leftarrow$ REFINE($G,\mathcal{P}$)
5:      **return** $\mathcal{P}$

6: **procedure** REFINE($G,\mathcal{P}$)
7:      **denote** $\mathcal{P} = \langle P_1, P_2 \ldots P_k \rangle$
8:      **for** $P_i \in \mathcal{P}$ **do**
9:          replace P_i by its partitioning $\langle U_1, \ldots U_k \rangle$ s.t $\forall u, v \in P_i$:
10:          $u, v \in U_j$ iff $\deg_G(u, \mathcal{P}) = \deg_G(v, \mathcal{P})$
11:      **return** $\mathcal{P}$

Example 2. Recall the graph described in Example 1. The nauty algorithm, starting with $\mathcal{P}^0 = \langle \{v_1, \ldots, v_5\} \rangle$, first distinguishes vertices by degree, obtaining the partitioning $\mathcal{P}^1 = \langle \{v_1, v_2, v_3\}, \{v_4\}, \{v_5\} \rangle$ as in Example 1. Observe that $\deg(v_1, \mathcal{P}^1) = \langle 2, 0, 0 \rangle$ and $\deg(v_2, \mathcal{P}^1) = \deg(v_3, \mathcal{P}^1) = \langle 1, 1, 0 \rangle$. Hence, in $\mathcal{P}^2 = \langle \{v_1\}, \{v_2, v_3\}, \{v_4\}, \{v_5\} \rangle$, the vertex v_1 is separated from v_2 and v_3. The partitioning $\mathcal{P}^2$ is equitable – $\deg(v_2, \mathcal{P}^2) = \deg(v_3, \mathcal{P}^2) = \langle 1, 0, 1, 0 \rangle$.   □
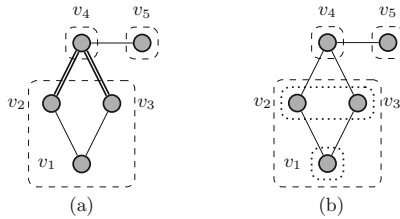
Fig. 2. The graph from Fig. 1 with (a) its partitioning $\mathcal{P}^1$ of vertices by degree, and (b) its refined partitioning $\mathcal{P}^2$. The partitioning in (b) is equitable.

Note that the order in which possible refinements are applied will not affect the *composition* of the resulting partitioning, but may change the *order* of parts. Any such order is acceptable, but it must be uniquely determined. In the following, we shall assume that at each step $i > 0$, listing the variables in $\mathcal{P}^i$ from left to right as $\langle v_{j_1}, \ldots, v_{j_n} \rangle$ then the sequence of vectors, $\deg(v_{j_1}, \mathcal{P}^{i-1}), \ldots, \deg(v_{j_n}, \mathcal{P}^{i-1})$ is sorted in decreasing lexicographic order.

If the partition $\mathcal{P}^{i-1}$ is derived from a structural property, then the composition and order of its parts must be invariant under permutation. So then is each vector $\deg(v_j, \mathcal{P}^{i-1})$, as permuting vertices within a part has no effect on the degrees – thus $\mathcal{P}^i$ is also structural.

Phase Two. Searching for a canonical representation: In the following we denote by $\mathcal{P}$ the equitable partitioning resulting from phase one. If $\mathcal{P}$ is *discrete*, then a canonical labeling of vertices has been established. However, this is rarely the case. Indeed, for regular graphs all vertices have the same degree, and are thus indistinguishable. In search for a canonical representation, nauty artificially selects some vertex in a non-singleton set $P \in \mathcal{P}$ to be made distinct from the other vertices of P. However, as these vertices are thus far indistinguishable, this cannot be done in a label-invariant fashion. So, each vertex in P is tentatively selected in turn, and a candidate discrete partitioning recursively constructed for each. The *canonical labeling* is then the candidate partitioning which yields the smallest graph under some total ordering.

We do not elaborate on the details of how this search is made as efficient as possible in nauty as the encodings introduced in this paper will take an alternative approach to search for a canonical representation given a partitioning $\mathcal{P}$.

2.3   Graph Search Problems and Breaking Symmetry

Graph search problems are about the search for a graph that satisfies certain properties. We will focus on properties that relate to the structure of the graph that ignore the particular names of the vertices. So if G is a solution to a graph search problem, then so is any G' that is isomorphic to G. More formally, an n-vertex graph search problem is a predicate, $\varphi(A)$, on an $n \times n$ matrix A of Boolean variables which is closed under isomorphism. A solution to $\varphi(A)$ is a satisfying assignment of the conjunction $\varphi(A) \wedge adj^n(A)$ where $adj^n(A)$ constrains A to be an $n \times n$ adjacency matrix. In Constraint (1), the left conjuct states that there are no self loops, and the right conjunct, that the edges are undirected.

$$adj^n(A) = \bigwedge \left\{ \, \neg A_{i,i} \, \middle| \, 1 \leq i \leq n \, \right\} \; \wedge \; \bigwedge \left\{ \, A_{i,j} \leftrightarrow A_{j,i} \, \middle| \, 1 \leq i < j \leq n \, \right\} \quad (1)$$

The set of solutions of graph search problem φ is denoted $sol(\varphi)$ and to make the variables explicit we write $sol(\varphi(A))$. Viewing $sol(\varphi)$ as a set of graphs, note that $sol(true) = \mathcal{G}_n$. The set $sol(\varphi)$ may include many isomorphic graphs; we write $sol_{\approx}(\varphi)$ to denote the set of solutions modulo graph isomorphism.

Example 3. The n vertex graph search problem $\varphi_{tm(n)}(G)$ is about the search for a Total Magic (TM) graph with n vertices. A graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, is TM if there exist a one-to-one labeling $\lambda : V \cup E \to \left\{ 1, \ldots, n + m \right\}$ and two integer values h, k which satisfy the constraints below. The graph is modeled as an $n \times n$ adjacency matrix A of Boolean variables. The edges are the unknown, hence m is unknown. The labeling is modeled as a length n vector λ^V of integer variables for the vertices, and an $n \times n$ matrix λ^E of integer variables for the edges. Note that both A and λ^E are symmetric. Let $M = n(n-1)/2$ be the maximum number of edges. Values in λ^V are between 1 and $n + M$. Values in λ^E are between 0 and $n + M$ where 0 is the value for "non-edges": $\lambda^E_{i,j}$ is zero if and only if $A_{i,j}$ is false.

Constraint (2) enforces that A is an adjacency matrix, and that m is the number of edges in the graph. Constraint (3) enforces that node labels are between

1 and $n + M$, that edge labels are between 0 and $n + M$ and are non-zero if the edge exists, and λ^E is symmetric. Constraint (4) enforces that nodes and edges are labeled differently (the λ_{ij}^E with label 0 are non-existing edges), and that the maximum label used is $n + m$, hence there is a bijection from vertices and edges to $\{1, \ldots, n + m\}$. We use++ to denote vector concatenation. Constraint (5) ensures the sum of the labels of each edge and its endpoints is k and the sum of the labels of each node and its incident edges is h.

$$adj^n(A) \wedge m = \sum_{i<j} A_{ij} \quad (2) \qquad \bigwedge_{1 \le i < j \le n} \left(\begin{array}{c} (1 < \lambda_i^V \le n+M) \wedge (0 \le \lambda_{ij}^E \le n+M) \\ \wedge\ (\lambda_{ij}^E > 0 \leftrightarrow A_{ij}) \wedge (\lambda_{ij}^E = \lambda_{ji}^E) \end{array} \right) \quad (3)$$

$$\mathsf{alldifferent\_except\_0}(\lambda^V\!\!+\!\!+\left[\lambda_{ij}^E | i < j\right]) \quad \wedge \quad \max(\lambda^V\!\!+\!\!+\lambda^E) = n + m \qquad (4)$$

$$\bigwedge_{i<j} A_{ij} \rightarrow (\lambda_i^V + \lambda_j^V + \lambda_{ij}^E = k) \quad \bigwedge \quad \bigwedge_{i \in V} (\lambda_i^V + \sum_{j \in V} \lambda_{ij}^E) = h \qquad (5)$$

There are only 6 TM graphs with up to 9 vertices and there exist no 10–11 vertex TM graphs [2]. The only known TM graphs, with >11 vertices, are composed of an odd number of triangles, or of an even number of triangles with a path of length 2. It is unknown if there exist other TM graphs with >11 vertices. □

Example 4. Several interesting relaxations of Total Magic graphs weaken the TM conditions. A graph is TM modulo p [2] if we replace the magic conditions with equality modulo p, that is replace Eq. (5) by

$$\bigwedge_{i<j} A_{ij} \rightarrow (\lambda_i^V + \lambda_j^V + \lambda_{ij}^E) \equiv k \mod p \quad \bigwedge \quad \bigwedge_{i \in V} (\lambda_i^V + \sum_{j \in V} \lambda_{ij}^E) \equiv h \mod p \quad (5')$$

We are often interested in finding graphs which are TM modulo several radices: $p_1, p_2, \ldots, p_k$. □

Solutions of a graph search problem are closed under permutations of the vertices. When solving graph search problems, it is essential to restrict the search space to break the symmetry between isomorphic solutions. Ideally, we would like to restrict the space to canonical representations.

Note however that we face a different problem to the methods described in Sect. 2.2. Canonicalization methods such as nauty take a *fixed* graph G, and compute some canonical representation $\mathsf{can}(G)$. Here, we must find some unknown graph satisfying φ, but wish to restrict our search to canonical representatives – that is, we wish to only accept graphs satisfying $G = \mathsf{can}(G)$.

A *symmetry break* is a predicate $\sigma(A)$ which is satisfied by at least one graph in each isomorphism class; a *complete* symmetry break is satisfied by exactly one graph in each equivalence class. A *canonizing* predicate, with respect to a total order $\preceq$ on graphs, is satisfied by exactly the set of minimal representations under $\preceq$. We shall use $sol_{\varphi}^{\sigma}(A)$ to denote the set of solutions to $\varphi(A)$ which satisfy the symmetry breaking predicate $\sigma(A)$.

Example 5. The following is a complete symmetry break, and a canonizing predicate with respect to $\preceq_{lex}$. It constrains A to be minimal with respect to all permutations of A.

$$\sigma_{\mathsf{clex}}(A) = \bigwedge_{\pi \in S_n} A \preceq_{lex} \pi(A) \tag{6}$$

Unfortunately the set S_n is prohibitively large, so this predicate is not at all practical. □

Example 6. The following is a partial symmetry break, introduced in [18]. It is a relaxation of σ_{clex}. It constrains A to be minimal with respect to all those permutations of A which swap a pair of elements.

$$\sigma_{plex}(A) = \bigwedge_{1 \le i < j \le n} A \preceq_{lex} \pi_{i,j}(A)$$

In practice this breaks many symmetries, and is of manageable size, and hence is often practically useful. □

In the following let $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ be an unknown partitioning of the vertices $V = \{1, \ldots, n\}$ of a graph expressed in terms of integer variables (so, when $p_i = p_j$ then vertices i and j are in the same part). We will make use of the following predicates:

The predicate $\mathsf{mono}(\mathcal{P})$ specifies that $\mathcal{P}$, represents a non-increasing sequence of values.

$$\mathsf{mono}(\mathcal{P}) = \bigwedge_{i=1}^{n-1} \mathcal{P}(i) \ge \mathcal{P}(i+1)$$

The predicate $\mathsf{plex}(A, \mathcal{P})$ specifies that an adjacency matrix A is minimal with respect to permutations that swap pairs of vertices in the same part of $\mathcal{P}$.

$$\mathsf{plex}(A, \mathcal{P}) = \bigwedge_{1 \le i < j \le n} \mathcal{P}(i) = \mathcal{P}(j) \rightarrow A \preceq_{lex} \pi_{i,j}(A)$$

The predicate $\mathsf{clex}(A, \mathcal{P})$ specifies that an adjacency matrix A is minimal with respect to permutations that preserve the partitioning $\mathcal{P}$.

$$\mathsf{clex}(A, \mathcal{P}) = \bigwedge_{\pi \in S_n} \pi(\mathcal{P}) = \mathcal{P} \rightarrow A \preceq_{lex} \pi(A) \tag{7}$$

Note that $\mathsf{plex}(A, \mathcal{P})$ and $\mathsf{clex}(A, \mathcal{P})$ are not symmetry breaks unless we also constrain A to have a structural property with the corresponding partitioning $\mathcal{P}$. We illustrate this in the following example.

Example 7. Consider a structural property μ and a predicate $\mu(A, \mathcal{P})$ which encodes that A is an $n \times n$ adjacency matrix (of Boolean variables) and $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ is a vector of integer variables such that $p_i = \mu_A(i)$. For instance, when $\mu = \mathsf{deg}$ we have

$$\mathsf{deg}(A, \mathcal{P}) = \bigwedge_{1 \le i \le n} \mathcal{P}(i) = \Sigma A_i$$

The following are respectively partial and complete symmetry breaks:

$$\sigma_\mu^{plex}(A) = \exists \mathcal{P}. \ \mu(A, \mathcal{P}) \wedge \mathsf{mono}(\mathcal{P}) \wedge \mathsf{plex}(A, \mathcal{P})$$

$$\sigma_\mu^{clex}(A) = \exists \mathcal{P}. \ \mu(A, \mathcal{P}) \wedge \mathsf{mono}(\mathcal{P}) \wedge \mathsf{clex}(A, \mathcal{P}) \qquad\qquad \square$$

3   The Nauty Encoding

In this section we describe a SAT encoding to break symmetries in graph search problems inspired by the way that nauty is applied to map a given graph to a canonical representation. We introduce a complete symmetry breaking predicate, $\sigma_{\mathsf{nauty}(k)}$, which similar to the nauty algorithm consists of two "phases" and takes the form:

$$\sigma_{\mathsf{nauty}(k)}(A) = \exists \mathcal{P}.\ \sigma_{\mathsf{nauty}(k)}^{\mathsf{phase}_1}(A, \mathcal{P}) \wedge \sigma_{\mathsf{nauty}(k)}^{\mathsf{phase}_2}(A, \mathcal{P})$$

The predicate $\sigma_{\mathsf{nauty}(k)}^{\mathsf{phase}_1}(A, \mathcal{P})$ accepts pairs consisting of an adjacency matrix A and a partitioning $\mathcal{P}$ such that executing the first phase of the nauty algorithm with k iterations on A results in the partitioning represented by $\mathcal{P}$, and the vertex order of A respects that partitioning. It further restricts A applying $\mathsf{plex}(A, \mathcal{P})$. The predicate $\sigma_{\mathsf{nauty}(k)}^{\mathsf{phase}_2}(A, \mathcal{P})$ accepts a pair $(A, \mathcal{P})$ if A is minimal in the class of graphs isomorphic to A which preserve the structural information in $\mathcal{P}$. The predicate $\sigma_{\mathsf{nauty}(k)}(A)$ accepts canonical adjacency matrices with respect to the structural information derived in the first phase of the nauty algorithm (k iterations). It is a complete symmetry break.

The essential difference between the encoding, $\sigma_{\mathsf{nauty}(k)}(A)$, and the nauty algorithm presented in Sect. 2.2 is that the nauty algorithm performs on a given graph where as the encoding, $\sigma_{\mathsf{nauty}(k)}(A)$, specifies constraints on an unknown graph A, restricting solutions for A to be canonical.

A partitioning $\mathcal{P}$ is represented as a vector $\langle p_1, \ldots, p_n \rangle$ of integer variables such that vertices v_i and v_j are in the same part if and only if $p_i = p_j$. When $\mathcal{P}$ is constrained to be monotone ($i < j \rightarrow \mathcal{P}(i) \leq \mathcal{P}(j)$) it may alternately be represented as a vector Δ of $n - 1$ of Boolean variables, such that $\Delta_i \leftrightarrow \mathcal{P}(i) < \mathcal{P}(i+1)$. Then v_i and v_j are in the same part if and only if $\Delta_i = \Delta_{i+1} = \ldots = \Delta_{j-1}$. The Boolean representation is more compact and performs better in our applications. Therefore, the nauty encoding is presented using the Boolean representation. Under the Boolean encoding, the predicate plex becomes:

$$\mathsf{plex}(A, \Delta) = \bigwedge_{1 \leq i < j \leq n} (\bigwedge_{i \leq k < j} \Delta_k) \rightarrow A \preceq_{\mathsf{lex}} \pi_{i,j}(A)$$

3.1   Encoding the First Phase of Nauty

To encode $\sigma_{\mathsf{nauty}(k)}^{\mathsf{phase}_1}$, we emulate the iterative refinement of partitionings. Let A be an $n \times n$ adjacency matrix and let Δ^i be the Boolean representation of the partitioning at step i of the nauty algorithm. We define a predicate $refine(\Delta^i, A, \Delta^{i+1})$ that specifies the partitioning, Δ^{i+1} at the next step of the algorithm. We then specify the predicate $\sigma_{\mathsf{nauty}(k)}^{\mathsf{phase}_1}$ as an iteration of this refinement, starting from the initial partition $\Delta^0 = \langle 0, \ldots, 0 \rangle$.

$$\sigma_{\mathsf{nauty}(k)}^{\mathsf{phase}_1}(A, \Delta) = iterate_k(\Delta^0, A, \Delta)\ \wedge\ \mathsf{plex}(A, \Delta)$$

$$iterate_k(\Delta, A, \Delta') = \begin{cases} \exists \Delta''.\ refine(\Delta, A, \Delta'') \wedge iterate_{k-1}(\Delta'', A, \Delta') & \text{if } k > 0 \\ \Delta = \Delta' & \text{if } k \leq 0 \end{cases}$$

As the graph is a "variable" (not given), we do not know in advance how many iterations are required to reach a fixpoint with respect to the structural information. However, as each non-trivial refinement must split some equivalence class, this process must reach a fixpoint after, at most, n iterations.

To facilitate the formal specification of the predicate $refine(\Delta, A, \Delta')$ we first introduce several "helper" predicates.

The predicate $P_{\leq}(\Delta, L)$: To compute structural information, it will be useful to identify those vertices appearing in parts *up to* the part containing some vertex v. The Boolean matrix L encodes this information.

$$P_{\leq}(\Delta, L) = \bigwedge_{1 \leq i,j \leq n} \begin{cases} L_{i,j} & \textbf{if } j \leq i \\ L_{i,j} \leftrightarrow L_{i,j-1} \wedge \neg\Delta_j & \textbf{otherwise} \end{cases}$$

The predicate $\mathsf{lex}(\Delta, M)$: This predicate specifies that the n rows of matrix M are non-increasing in the lexicographic order following the length $n-1$ vector Δ. The intention is that Δ specifies a partitioning.

$$\mathsf{lex}(\Delta, M) = \bigwedge_{1 \leq i < n} (\neg\Delta_i \to M_i \succeq_{lex} M_{i+1})$$

The predicate $\mathsf{deg}(A, \Delta, M)$: This predicate defines a relationship between: an $n \times n$ matrix, A, of Boolean variables (representing an unknown graph), a length $n-1$ vector, Δ, of Boolean variables (representing a partitioning of the vertices in A), and an $n \times n$ matrix, M, of integer variables such that $M_{i,j}$ represents the number of edges from vertex i to vertices in or before the part number containing vertex j. The rows of M are ordered lexicographically within each component of $\mathcal{P}$. The predicate is specified as:

$$\mathsf{deg}(A, \Delta, M) = \bigwedge_{1 \leq i,j \leq n} \left(M_{i,j} = \sum_{k=1}^{n} A_{i,k} \wedge L_{j,k} \right) \wedge \mathsf{lex}(\Delta, M)$$

The predicate $refine(\Delta, A, \Delta')$: This predicate states that Δ' is a refinement of the partitioning Δ of graph A obtained from a single iteration of partition refinement. The matrix M represents structural information of the vertices with respect to the partitioning Δ. Vertices are distinguished in the refinement Δ': either because they were already distinguished in Δ, or else because they are distinguished by the corresponding structural information in M.

$$refine(\Delta, A, \Delta') = \left(\exists M.\, \mathsf{deg}(A, \Delta, M) \wedge \bigwedge_{1 \leq i < n} (\Delta'_i \leftrightarrow \Delta_i \vee M_i \succ_{lex} M_{i+1}) \right)$$

The encoding of predicate $\sigma_{\mathsf{nauty}(k)}^{\mathsf{phase}_1}$ is polynomial in the number of both clauses and variables. The dominating component of *refine* is deg, which introduces $O(|V|^2)$ order-encoded integer variables whose definitions are sums of Booleans, which have standard polynomial-size encodings (e.g. [13]).

Table 1. Enumerating graphs using $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(k)}$. Column "sat" is Clasp solving time (sec).

n	$|\mathcal{G}_n^{\approx}|$	$\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(0)}$ == σ_{plex} [18]				$\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(1)}$				$\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(2)}$			
		Cls	Vars	Sat	Sols	Cls	Vars	Sat	Sols	Cls	Vars	Sat	Sols
3	4	2	3	0.00	4	76	21	0.00	4	357	91	0.00	4
4	11	20	10	0.00	11	243	56	0.00	11	1142	279	0.00	11
5	34	70	24	0.00	43	551	110	0.01	34	2618	610	0.01	34
6	156	165	48	0.00	276	1048	192	0.01	158	5113	1165	0.06	156
7	1,044	320	85	0.02	3,158	1765	301	0.06	1,141	8870	1969	0.45	1,048
8	12,346	550	138	0.32	66,595	2741	440	0.59	14,745	14196	3067	6.63	12,642
9	274,668	870	210	12.13	2,587,488	4015	612	12.51	355,294	21422	4504	159.40	284,041
10	12,005,168	1295	304	1035.51	184,192,329	5646	830	819.93	16,255,967	31123	6435	6511.07	12,442,095

Table 1 illustrates the impact of structural information when breaking symmetries and enumerating the graphs obtained with n vertices. The column headed by $|\mathcal{G}_n^{\approx}|$ indicates the number of non-isomorphic graphs with n vertices. These numbers correspond to sequence A000088 of the OEIS [19]. The next columns, in groups of 4, are headed by $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(k)}$ for $0 \leq k \leq 2$. Each such foursome details the size of the SAT encoding (number of clauses and variables), sat solving time (for all solutions in seconds), and the number of solutions found. When $k = 0$ there is no structural information and the encoding corresponds to the one introduced in [18]. When $k = 1$, the nodes of the graph are partitioned according to degree information. When $k > 0$, the symmetry breaks are more refined than the one introduced in [18]. Notice that as we add structural information in the encoding (as k increases), the number of graphs decreases. For example, when $n = 10$, using $k = 0$ there are circa 184 million solutions, when $k = 1$, circa 16 million, and $k = 2$, circa 12 million (close to the true number $|\mathcal{G}_{10}^{\approx}|$). Note that as we add structural information, the cost of the solving time increases considerably. In the following we will show how to counter this increase.

Table 2 summarizes an application of symmetry breaking with the first phase of nauty to search for all TM graphs (modulo 2,3) (see Example 4). The columns, in groups of 4, are headed by $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(k)}$ for $0 \leq k \leq 3$. Each such foursome details the size of the SAT encoding (number of clauses and variables), sat solving time in seconds unless indicated otherwise, and the number of solutions generated. The table illustrates the high cost of the nauty encoding: we can solve up to $n = 9$, and then only for $k = 1$. We will come back to resolve this problem below by decomposing the instances to consider given nauty partitionings.

3.2   Encoding the Second Phase of Nauty

We present a symmetry break predicate, $\sigma^{\mathsf{phase}_2}_{\mathsf{nauty}(k)}$, which eliminates isomorphic graphs that have not been ruled out by the first phase predicate, $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(k)}$. In the second phase of nauty, the graph G is given, and so is the partitioning $\mathcal{P}$, from its first phase computation. In our case, we seek a predicate that states that an unknown graph, G, is canonical. The search strategy applied in nauty is not easily modeled as a propositional formula when G is unknown and so we

Table 2. TM (modulo 2,3) graphs using $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(k)}$ (48 h timeout using Clasp).

| | $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(0)}$ == σ_{plex} [18] | | | | $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(1)}$ | | | | $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(2)}$ | | | | $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(3)}$ | | | |
n	Cls	Vars	Sat	Sols	Cls	Vars	Sat	Sols	Cls	Vars	Sat	Sols	Cls	Vars	Sat	Sols
3	791	216	0.00	3	852	231	0.00	3	1139	290	0.00	3	1426	349	0.00	3
4	1645	422	0.01	4	1817	459	0.02	4	2702	626	0.02	4	3587	793	0.02	4
5	3024	717	0.16	16	3379	785	0.13	13	5398	1139	0.15	13	7417	1493	0.29	13
6	4973	1105	2.03	60	5606	1219	1.49	39	9574	1896	1.75	39	13542	2573	3.89	39
7	7705	1590	36.06	426	8715	1761	24.27	179	15658	2908	72.60	171	22601	4055	140.68	171
8	11438	2180	6891.22	7087	12936	2419	1795.15	1647	24154	4217	9799.19	1447	35372	6015	15828.39	1430
9	16275	2881	T.O	-	18385	3200	39.71 h	36984	35456	5844	T.O	-	52527	8488	T.O	-

introduce an alternative approach, assuming that the partitioning $\mathcal{P}$, from the first phase, is given. As a starting point, consider $\mathsf{clex}(A, \mathcal{P})$ given as Eq. (7). This predicate provides a complete symmetry break when combined with $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}}$. However, its implementation is inefficient as the encoding must consider each of the permutations in S_n which preserve $\mathcal{P}$, and their number may be huge.

In [12] the authors show that complete symmetry breaks for graph isomorphism with n vertices can be obtained using only a small fraction of the required $n!$ permutations. For example, [12] reports that a complete symmetry break for $n = 10$ vertices involves only 7853 permutations whereas the complete symmetry break $\sigma_{\mathsf{clex}}(A)$ introduced as Eq. (6) in Example 5 involves $10! = 3{,}628{,}800$ permutations. In this paper we enhance the approach of [12] to consider partitionings expressed by encodings of the first stage of the nauty algorithm.

A common approach for improving performance of combinatorial existence checking is decomposition: splitting the problem into a manageable set of disjoint subproblems, each of which can (ideally) be solved more easily. We can decompose a graph search problem with respect to the partitioning inferred by $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(k)}$. This results in a decomposition to 2^{n-1} subproblems, one for each partitioning.

To compute a concise and complete symmetry break for a partitioning $\mathcal{P}$ corresponding to the first phase $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(k)}$ we apply the same approach as advocated in [12], but restricted to break symmetries on graphs which have the structural information $\mathcal{P}$ after k iterations of the nauty first phase algorithm. This computation is performed by application of Algorithm 2 where we denote: (a) $S_n^{\mathcal{P}}$ the set of permutations that preserve a partitioning $\mathcal{P}$, and (b) for $\Pi \subseteq S_n$, $\min_\Pi(G) = \bigwedge \left\{ \, G \preceq \pi(G) \, \middle| \, \pi \in \Pi \, \right\}$.

For a given partitioning $\mathcal{P}$ and value k, Algorithm 2 starts with an empty set of permutations Π and iterates adding permutations as long as the condition in Line 3 holds. The condition seeks a pair (G, π) such that $\pi(G) \preceq G$, where $G \in sol(\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(k)}(A, \mathcal{P}))$ and π preserves $\mathcal{P}$. Such a graph violates $\mathsf{clex}(\mathcal{P}, G)$ and hence π is added to Π. The implementation performs this check by invoking a SAT solver, the same as in [12].

We say that Π is redundant if there exists π such that for all G, $\min_\Pi(G) \leftrightarrow \min_{\Pi \setminus \{\pi\}}(G)$. The set computed by the while-loop at Line 3 may be non-minimal, as an existing permutation may become redundant in view of

Algorithm 2. Compute Canonizing Set

```
1: procedure COMPUTE-CANONIZING-SET(P, k)
2:     Init: Π = ∅
3:     while ∃(G, π) ∈ (sol(σ^{phase_1}_{nauty(k)}(A, P)), S_n^P) s.t min_Π(G) ∧ π(G) ≺ G do
4:         Π = Π ∪ {π}
5:     for each π ∈ Π do
6:         if ∀G ∈ sol(σ^{phase_1}_{nauty(k)}(A, P)): min_{Π\{π}}(G) ⇒ G ⪯ π(G) then
7:             Π = Π \ {π}
8:     return Π
```

permutations added later. Thus the algorithm then iterates to remove redundant permutations applying the for-loop at Line 5.

Table 3 is about computing the permutations to make the nauty partitionings complete (phase2). For each n (number of vertices), we indicate ("parts") the number of partitions. We detail separately the cost of computing the permutations for the regular partitioning ("regular part") where degree-based structural information has no impact. These have the most permutations and are the most costly to compute. Then for $\sigma_{\mathsf{nauty}(1)}$, $\sigma_{\mathsf{nauty}(2)}$ and $\sigma_{\mathsf{nauty}(3)}$ we detail the number of permutations and the time to compute them using Algorithm 2. For the number of permutations we detail x/y where x is the largest number of permutations computed for a single partitioning, and y is the total number. For the time we detail x/y where x is the longest time to compute for a single partitioning and y is the total time.

Notice how the required number of permutations decreases as structural information is added (k increases). For $n = 10$ we have 8608 permutations with $k = 1$, 3703 with $k = 2$, and 1497 when $k = 3$. Recall that without structural information 7853 permutations were required [12], and computation beyond 10 vertices was not possible. Moreover note that because of the decomposition to partitionings we require no more than 37 permutations to break all symmetries on 10 vertices given a partitioning derived from $\sigma^{\mathsf{phase}_1}_{\mathsf{nauty}(3)}$. The permutations computed here provide complete symmetry breaks for any graph search problem with up to 12 vertices.

Table 3. Canonizing sets per partitioning. Time in seconds except under the line (for $n = 11, 12$) where in hours. Timeout is 48 h using Glucose.

n	Parts	Regular part		$\sigma_{\mathsf{nauty}(1)}$		$\sigma_{\mathsf{nauty}(2)}$		$\sigma_{\mathsf{nauty}(3)}$	
		Perms	Time	Perms	Time	Perms	Time	Perms	Time
6	32	0	0.22	1/2	0.09/1.73	0/0	0.12/2.60	0/0	0.18/3.87
7	64	2	0.14	5/49	0.25/6.24	1/2	0.24/10.75	0/0	0.37/15.68
8	128	12	1.22	18/330	2.49/35.11	5/93	1.29/62.21	6/40	2.33/82.52
9	256	20	4.54	44/1875	22.38/303.19	13/640	4.16/400.65	14/225	7.88/512.03
10	512	144	447.48	215/8608	750.2/4649.30	51/3703	67.16/3309.17	37/1497	95.83/4576.36
11	1024	346	0.84	1030/44521	8.36/44.9	169/16391	0.22/9.62	171/6718	0.19/12.91
12	2048	[a]13139	16.32	-	T.O	718/77158	3.08/99.44	577/33182	2.22/116.2

[a] To stay within the time-out, the computation of these permutations omits the removal of redundant permutations, skipping the for-loop at Line 5 in Algorithm 2.

Table 4. Enumerating TM (modulo 2,3) graphs using $\sigma_{\mathsf{nauty}(k)}$ complete symmetry breaks. Time in seconds except under the line (for $n = 9, 10$) where in hours. Column 1 computed with Clasp, column 2,3 computed with Glucose. (120 h timeout).

n	σ^*_{clex}			$\sigma_{\mathsf{nauty}(1)}$				$\sigma_{\mathsf{nauty}(2)}$				sols
	Cls	Vars	Sat	Inst.	Cls	Vars	Sat	Inst.	Cls	Vars	Sat	
3	791	216	0.00	4	346	80	0.00/0.00	4	346	80	0.00/0.00	3
4	1640	421	0.02	11	935	218	0.00/0.02	11	1053	233	0.00/0.02	4
5	3179	748	0.13	31	1783	409	0.01/0.18	33	2163	466	0.01/0.21	13
6	5093	1129	1.03	102	3088	699	0.06/1.52	143	4254	882	0.04/1.92	39
7	8710	1791	14.49	342	4794	1060	1.07/26.19	755	7293	1462	0.33/28.24	171
8	21633	4219	304.20	1213	7091	1524	18.16/766.32	4817	11503	2220	5.87/678.68	1425
9	105030	20632	11.91	4361	10079	2104	0.17/11.73	32883	16863	3140	0.07/12.71	29415
10	1428281	284565	T.O	16016	14466	2925	T.O	223554	23572	4233	114.51/2215.12	1099398

A first attempt to enumerate all TM graphs modulo 2,3 using the complete symmetry breaks $\sigma_{\mathsf{nauty}(k)}$ per partitioning failed. The instances are simply too hard. To this end, we took a second approach where the encoding for each partitioning was enhanced with additional information on the degree sequences of solutions. Namely, the implementation considers for each partition all possible relevant degree sequences and solve each instance separately.

Table 4 summarizes the results. For each n we detail the results obtained using the complete symmetry breaks of [12] denoted by σ^*_{clex} (these symmetry breaks are equivalent to $\sigma_{clex}(A)$ but are much compact), and then the results for $\sigma_{\mathsf{nauty}(k)}$ with $k = 1, 2$. Here we detail the total number of instances (on all partitionings). For time we detail x/y where x is the time for the hardest instance and y is the total time. The number of solutions is the same as this is, in all three cases, the number of canonical solutions.

4   Conclusion

This paper presents polynomial-size static symmetry breaking predicates which encode structural properties in the same way that nauty exploits information when it maps graphs to their canonical representations. These structural breaks apply to strengthen existing incomplete symmetry-breaking predicates, and can be extended into complete symmetry breaks. These structural properties also yield a natural strategy for problem decomposition. We have described a SAT encoding for the structural symmetry-breaking predicates, and applied these to compute compact, complete symmetry breaks for graphs of up to 12 vertices. We also demonstrated the effectiveness of these structural techniques in accelerating the enumeration of TM graphs modulo 2, 3. Ongoing work focuses on an encoding that exploits on richer structural properties than the current focus on vertex degree. As described in [17], this is expected to improve the situation when breaking symmetries on regular graphs. We also plan to apply the same technique to compute sets of permutations with which to break symmetries for a given graph search problem. We expect to then be able to apply the technique for larger instances than those we can do now.

References

1. Read, R.C.: Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations. Ann. Discrete Math. **2**, 107–120 (1978)
2. Jäger, G., Arnold, F.: SAT and IP based algorithms for magic labeling including a complete search for total magic labelings. J. Discrete Algorithms **31**, 87–103 (2015)
3. Puget, J.: On the satisfiability of symmetrical constrained satisfaction problems. In: Methodologies for Intelligent Systems, 7th International Symposium, ISMIS 1993, Trondheim, Norway, 15–18 June 1993, Proceedings, pp. 350–361 (1993)
4. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In: Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR 1996), Cambridge, Massachusetts, USA, 5–8 November 1996, pp. 148–159 (1996)
5. Shlyakhter, I.: Generating effective symmetry-breaking predicates for search problems. Discrete Appl. Math. **155**(12), 1539–1548 (2007)
6. Walsh, T.: General symmetry breaking constraints. In: Principles and Practice of Constraint Programming - Cp. 2006, 12th International Conference, Cp. 2006, Nantes, France, 25–29 September 2006, Proceedings, pp. 650–664 (2006)
7. Gent, I.P., Smith, B.M.: Symmetry breaking in constraint programming. In: Horn, W. (ed.) ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, 20–25 August 2000, pp. 599–603. IOS Press (2000)
8. Mears, C., de la Banda, G., Demoen, B., Wallace, M.: Lightweight dynamic symmetry breaking. Constraints **19**(3), 195–242 (2013)
9. McKay, B.D.: Practical graph isomorphism. Congressus Numerantium **30**, 45–87 (1981)
10. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments, SIAM, pp. 135–149 (2007)
11. Darga, P.T., Liffiton, M.H., Sakallah, K.A., Markov, I.L.: Exploiting structure in symmetry detection for CNF
12. Itzhakov, A., Codish, M.: Breaking symmetries in graph search with canonizing sets. Constraints **21**, 1–18 (2016)
13. Metodi, A., Codish, M., Stuckey, P.J.: Boolean equi-propagation for concise and efficient SAT encodings of combinatorial problems. J. Artif. Intell. Res. (JAIR) **46**, 303–341 (2013)
14. Audemard, G., Simon, L.: Glucose 4.0 SAT Solver. http://www.labri.fr/perso/lsimon/glucose/
15. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: from theory to practice. Artif. Intell. **187**, 52–89 (2012)
16. Erdös, P., Gallai, T.: Graphs with prescribed degrees of vertices (in Hungarian). Mat. Lapok **11**, 264–274 (1960). http://www.renyi.hu/~p_erdos/1961-05.pdf
17. McKay, B.D., Piperno, A.: Practical graph isomorphism II. J. Symbolic Comput. **60**, 94–112 (2014)
18. Codish, M., Miller, A., Prosser, P., Stuckey, P.J.: Breaking symmetries in graph representation. In: Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, IJCAI/AAAI (2013)
19. Sloane, N.J.A. (ed.): The On-Line Encyclopedia of Integer Sequences. https://oeis.org Accessed April 2016

Extending Broken Triangles and Enhanced Value-Merging

Martin C. Cooper[1], Achref El Mouelhi[2(✉)], and Cyril Terrioux[2]

[1] IRIT, University of Toulouse III, 31062 Toulouse, France
cooper@irit.fr
[2] Aix-Marseille Université, CNRS, ENSAM, Université de Toulon,
LSIS UMR 7296, 13397 Marseille, France
{achref.elmouelhi,cyril.terrioux}@lsis.org

Abstract. Broken triangles constitute an important concept not only for solving constraint satisfaction problems in polynomial time, but also for variable elimination or domain reduction by merging domain values. Specifically, for a given variable in a binary arc-consistent CSP, if no broken triangle occurs on any pair of values, then this variable can be eliminated while preserving satisfiability. More recently, it has been shown that even when this rule cannot be applied, it could be possible that for a given pair of values no broken triangle occurs. In this case, we can apply a domain-reduction operation which consists in merging these values while preserving satisfiability.

In this paper we show that under certain conditions, and even if there are some broken triangles on a pair of values, these values can be merged without changing the satisfiability of the instance. This allows us to define a stronger merging operation and a new tractable class of binary CSP instances. We report experimental trials on benchmark instances.

1 Introduction

Identifying tractable classes constitutes an important research goal in constraint programming. The *broken-triangle property* (BTP) defines a hybrid tractable class [6,7]. This class has some interesting characteristics, both from a theoretical and practical viewpoint: it generalises existing language-based and structural classes and is solved in polynomial time by the algorithm MAC which is omnipresent in constraint solvers [20]. Besides, many extensions of the broken-triangle property have led to the definition of new tractable classes [8,10,11,14,18,19]. Local versions of the BTP have also given rise to novel reduction operations for CSP instances. In particular, in arc-consistent binary CSP instance, if no broken triangle occurs on any pair of values in the domain of a variable, then this variable can be eliminated without changing the satisfiability of the instance [2]. Even when this variable-elimination rule cannot be applied, it can nevertheless happen that no broken triangle occurs on a particular pair of values. In this case, these two values can be merged into a single value without changing the satisfiability of the instance [5]. This domain-reduction

operation, known as BT-merging, was found to be applicable in diverse bench-mark domains, although extensive experimental trials would seem to indicate that it is not useful, in terms of total solving time, as a preprocessing operation in a general-purpose solver [4].

In the light of these results, in this paper we study a lighter version of BTP-merging which allows the presence of some broken triangles on the pair of values to be merged, thus giving rise to a stronger domain-reduction operation.

In the following section we recall basic definitions and notations used in the rest of the paper. In Sect. 3 we introduce a new generic rule, called m-wBTP, which allows us to merge two values even in the presence of some broken triangles. We then show in Sect. 4 that, for sufficiently large m, this rule is maximal. We go on to show, in Sect. 5, that this merging rule does not allow the elimination of variables. Nevertheless, in Sect. 6 we show that it does allow us to define a tractable class. We also compare m-wBTP with certain other generalisations of BTP, such as k-BTP [8] and WBTP [19]. In Sect. 7 we report experimental trials to evaluate the practical interest of 1-wBTP-merging.

2   Preliminaries

Constraint satisfaction problems (CSPs [17]) are at the heart of numerous applications in Artificial Intelligence and Operations Research. In this paper, we study only binary CSP instances, defined formally as follows:

Definition 1. *A **binary CSP instance** is a triple $I = (X, D, C)$, where $X = \{x_1, \ldots, x_n\}$ is a finite set of n **variables**, $D = \{D(x_1), \ldots, D(x_n)\}$ is a set of **domains** containing at most d **values**, a domain for each variable, and C is a set of **binary constraints**. Each constraint $C_{ij} \in C$ is a pair $(S(C_{ij}), R(C_{ij}))$ with:*

- *$S(C_{ij}) = \{x_i, x_j\} \subseteq X$, the **scope** of the constraint,*
- *$R(C_{ij}) \subseteq D(x_i) \times D(x_j)$, the **relation** specifying the compatibility of values.*

If the constraint C_{ij} is not defined in C, then we consider C_{ij} to be a universal constraint (i.e. such that $R(C_{ij}) = D(x_i) \times D(x_j)$).

The interaction between the values of each variable through the relations associated to constraints can be represented graphically by a *microstructure* graph [13]. The vertices of this graph are thus the variable-value pairs (x_i, v_i) $(v_i \in D(x_i))$ and the edges are the tuples authorized by the constraints (that is, there is an edge between the vertices (x_i, v_i) and (x_j, v_j) iff $(v_i, v_j) \in R(C_{ij})$). Given a binary instance I, deciding whether I has a *solution* (an assignment $(v_1, \ldots, v_n)$ such that $\forall i,\ v_i \in D(x_i)$ and $\forall i \neq j,\ (v_i, v_j) \in R(C_{ij})$), is well known to be NP-complete. However, by imposing some restrictions on the constraint scopes and/or relations, we can define *tractable* classes of instances which can be solved in polynomial time. The BTP (*Broken Triangle Property*) tractable class, is an important tractable class since it generalises certain previously known classes

based exclusively on properties of the constraint scopes or the constraint relations and has been the inspiration for a new branch of research on tractable classes of CSPs based on forbidden patterns [1,4,9,11,18]. The Broken Triangle Property imposes the *absence* of so-called broken triangles. Formally, BTP is defined as follows:

Definition 2 (Broken-Triangle Property [6,7]). *Let I be a binary CSP instance with a variable order $<$. A pair of values $v'_k, v''_k \in D(x_k)$ satisfies BTP if, for each pair of variables (x_i, x_j) such that $x_i < x_j < x_k$, $\forall v_i \in D(x_i)$, $\forall v_j \in D(x_j)$, if $(v_i, v_j) \in R(C_{ij})$, $(v_i, v'_k) \in R(C_{ik})$ and $(v_j, v''_k) \in R(C_{jk})$, then either $(v_i, v''_k) \in R(C_{ik})$, or $(v_j, v'_k) \in R(C_{jk})$.*

A variable x_k satisfies BTP if each pair of values in $D(x_k)$ satisfies BTP. An instance satisfies BTP if all its variables satisfy BTP.

This definition can be represented graphically in the microstructure of I as shown in Fig. 1. Throughout this paper, we represent an unauthorized assignment (a tuple which violates the constraint) either by a dashed line or by the absence of a line.
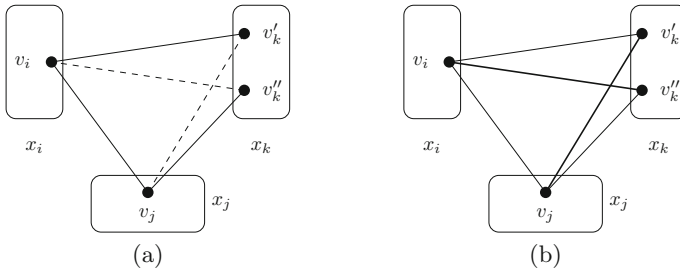
Fig. 1. (a) A broken triangle (v_i, v_j, v'_k, v''_k). (b) The assignments (v_i, v_j, v'_k, v''_k) do not form a broken triangle.

In Fig. 1(a), the CSP instance is not BTP relative to the order $x_i < x_j < x_k$ because the tuples (v_j, v'_k) and (v_i, v''_k) are not authorized. In this example, (v_i, v_j, v'_k, v''_k) constitute a *broken triangle* on the values v'_k and v''_k. Because of this broken triangle, we say that there is a broken triangle on x_k relative to x_i and x_j. On the other hand, if $(v_i, v''_k) \in R(C_{ik})$ or $(v_j, v'_k) \in R(C_{jk})$, as illustrated in Fig. 1(b), then the broken-triangle property is satisfied.

We now define the merging of domain values before recalling the merging operation based on BTP.

Definition 3 [4]. *Merging the values $v'_k, v''_k \in D(x_k)$ in a binary CSP instance I consists of replacing v'_k, v''_k in $D(x_k)$ by a new value v_k which is compatible with all values which are compatible with at least one of the values v'_k or v''_k. A **value-merging condition** is a polytime-verifiable property such that when it holds on a pair of values $v'_k, v''_k \in D(x_k)$, the CSP instance obtained after merging the values v'_k and v''_k is satisfiable if and only if I was satisfiable.*

In binary CSP instances, the absence of broken triangles on a pair of values is a valid value-merging condition [4]. For example, in Fig. 1(b), the values v'_k and v''_k are mergeable.

3   Weakly Broken Triangles

The absence of broken triangles on a pair of values allows them to be merged while preserving satisfiability. In this section, we show that it is possible to merge certain pairs of values even in the presence of some broken triangles. This idea was inspired by recent work by Naanaa [19] on a new extension of BTP. We call our new property m-wBTP: the parameter m defines the number of variables supporting the *weakly broken triangles*.

3.1   1-wBTP-Merging

We start with the simplest case ($m = 1$) based on a new concept called *weakly broken triangles* supported by one other variable.

Definition 4. *A pair of values $v'_k, v''_k \in D(x_k)$ satisfies* 1-**wBTP** *if for each broken triangle (v_i, v_j, v'_k, v''_k) with $v_i \in D(x_i)$ and $v_j \in D(x_j)$, there is at least one variable $x_\ell \in X \setminus \{x_i, x_j, x_k\}$ such that: $\forall\, v_\ell \in D(x_\ell)$ if $(v_i, v_\ell) \in R(C_{i\ell})$ and $(v_j, v_\ell) \in R(C_{j\ell})$ then*

- $(v'_k, v_\ell) \notin R(C_{k\ell})$ *and*
- $(v''_k, v_\ell) \notin R(C_{k\ell})$.

If this is the case, (v_i, v_j, v'_k, v''_k) is known as a **weakly broken triangle** *supported by the variable x_ℓ.*

This definition can be represented by the microstructure graph, as shown in Fig. 2. There is a broken triangle (v_i, v_j, v'_k, v''_k). Since for each value v_ℓ of the variable x_ℓ, v_ℓ is compatible with v_i and v_j and we have $(v'_k, v_\ell) \notin R(C_{k\ell})$ and $(v''_k, v_\ell) \notin R(C_{k\ell})$, this triangle is a weakly broken triangle supported by x_ℓ.

The notion of weakly broken triangles allows us to generalise BTP-merging.

Proposition 1. *In a binary CSP, merging two values $v'_k, v''_k \in D(x_k)$ which satisfy* 1-wBTP *does not change the satisfiability of an instance.*

Proof. Let I be the original instance and I' the new instance in which v'_k, v''_k have been merged into a new value v_k (which replaces v'_k, v''_k in $D(x_k)$). Clearly, if I is satisfiable then so is I'. Hence, it suffices to show that if I' has a solution s which assigns v_k to x_k, then I also has a solution.

Let s', s'' be two assignments which are identical to s except that s' assigns v'_k to x_k and s'' assigns v''_k to x_k. Suppose, for a contradiction, that neither s' nor s'' is a solution to I. Then there are two variables $x_i, x_j \in X \setminus \{x_k\}$ such that $(s(x_i), v'_k) \notin R(C_{ik})$ and $(s(x_j), v''_k) \notin R(C_{jk})$. Since s is a solution to I' assigning v_k to x_k, we must have $(s(x_i), v''_k) \in R(C_{ik})$ and $(s(x_j), v'_k) \in R(C_{jk})$.

Fig. 2. A triangle which is weakly broken since $(v_k', v_\ell) \notin R(C_{k\ell})$ and $(v_k'', v_\ell) \notin R(C_{k\ell})$.

Obviously, we also have $(s(x_i), s(x_j)) \in R(C_{ij})$ since s is a solution to I'. So $(s(x_i), s(x_j), v_k', v_k'')$ is a broken triangle in I.

By the definition of 1-wBTP, there is a variable $x_\ell \in X \setminus \{x_i, x_j, x_k\}$ such that $\forall v_\ell \in D(x_\ell)$ if $(s(x_i), v_\ell) \in R(C_{i\ell})$ and $(s(x_j), v_\ell) \in R(C_{j\ell})$ then

- $(v_k', v_\ell) \notin R(C_{k\ell})$ and
- $(v_k'', v_\ell) \notin R(C_{k\ell})$.

As $s(x_\ell)$ is compatible with $s(x_i)$ and $s(x_j)$, it cannot be compatible with either v_k' or v_k''. It follows that $s(x_\ell)$ is not compatible with v_k, which implies that s is not a solution to I'. But this contradicts our initial hypothesis. Thus, this merging rule preserves satisfiability. □
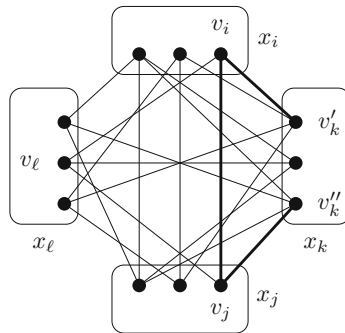
Fig. 3. A CSP instance in which all values are arc consistent (in bold, the weakly broken triangle).

At first sight, there appears to be an obvious link between this definition and arc consistency [16]. Indeed, imposing that the tuples (v_k', v_ℓ) and (v_k'', v_ℓ) are unauthorized seems to imply that the goal is to render the values v_k' and v_k'' arc-inconsistent. But the example in Fig. 3 shows that this is not always the case.

Indeed, although the two values $v'_k, v''_k \in D(x_k)$ in this figure satisfy 1-wBTP, establishing arc consistency deletes no values (and obviously no tuples). Thus, arc consistency does not delete the broken triangle (v_i, v_j, v'_k, v''_k).

3.2 m-wBTP-Merging

In Definition 4, thanks to the supporting variable(s) x_ℓ, merging values on which there are only weakly broken triangles leaves the satisfiability of the instance invariant. In terms of the microstructure, the variable x_ℓ prevents the creation of a new clique in the microstructure of size n (i.e. a new solution) which did not exist before merging. This principle can clearly be extended to m variables ($m \leq n - 3$).

An assignment $(v_{\ell_1}, \ldots, v_{\ell_m}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_m})$ is a **partial solution** if it satisfies all constraints C_{ij} such that $\{x_i, x_j\} \subseteq \{x_{\ell_1}, \ldots, x_{\ell_m}\}$.

Definition 5. *A pair of values $v'_k, v''_k \in D(x_k)$ satisfies m-**wBTP** where $m \leq n - 3$ if for each broken triangle (v_i, v_j, v'_k, v''_k) with $v_i \in D(x_i)$ and $v_j \in D(x_j)$, there is a set of $r \leq m$ **support variables** $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq X \setminus \{x_i, x_j, x_k\}$ such that for all $(v_{\ell_1}, \ldots, v_{\ell_r}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_r})$, if $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i, v_j)$ is a partial solution, then there is $\alpha \in \{1, \ldots, r\}$ such that $(v_{\ell_\alpha}, v'_k), (v_{\ell_\alpha}, v''_k) \notin R(C_{\ell_\alpha k})$. We say that x_{ℓ_α} is the **shield variable** for this partial solution.*

Figure 4 shows two configurations illustrating Definition 5. In the first, the pair of values $v'_k, v''_k \in D(x_k)$ satisfies 2-wBTP because for the unique partial solution $(v_{\ell_\sigma}, v_{\ell_\gamma}, v_i, v_j)$ we have $(v_{\ell_\sigma}, v'_k), (v_{\ell_\sigma}, v''_k) \notin R(C_{\ell_\sigma k})$. In the second, there is no partial solution on the set of variables $\{x_{\ell_\sigma}, x_{\ell_\gamma}, x_i, x_j\}$; thus $v'_k, v''_k \in D(x_k)$ trivially satisfies 2-wBTP.

(a)   (b)
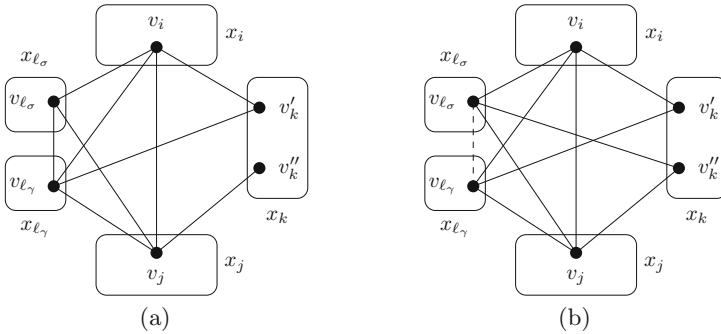
Fig. 4. Two different cases of two values v'_k and v''_k which satisfy 2-wBTP.

We now generalise Proposition 1 to the merging of values satisfying m-wBTP.

Proposition 2. *In a binary CSP, merging two values $v'_k, v''_k \in D(x_k)$ which satisfy m-wBTP does not change the satisfiability of an instance.*

Proof. Let I be the original instance and I' the new instance in which v'_k, v''_k have been merged into a new value v_k (which replaces v'_k, v''_k in $D(x_k)$). Clearly, if I is satisfiable then so is I'. Hence, it suffices to show that if I' has a solution s which assigns v_k to x_k, then I also has a solution.

Let s', s'' be two assignments which are identical to s except that s' assigns v'_k to x_k and s'' assigns v''_k to x_k. Suppose, for a contradiction, that neither s' nor s'' is a solution to I. Then there are two variables $x_i, x_j \in X \setminus \{x_k\}$ such that $(s(x_i), v'_k) \notin R(C_{ik})$ and $(s(x_j), v''_k) \notin R(C_{jk})$. Since s is a solution to I' assigning v_k to x_k, we must have $(s(x_i), v''_k) \in R(C_{ik})$ and $(s(x_j), v'_k) \in R(C_{jk})$. We also have $(s(x_i), s(x_j)) \in R(C_{ij})$ since s is a solution to I. So $(s(x_i), s(x_j), v'_k, v''_k)$ is a broken triangle in I.

The values v'_k and v''_k satisfy m-wBTP, so, by definition, there is a set of $r \leq m$ variables $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq X \setminus \{x_i, x_j, x_k\}$ such that for all $(v_{\ell_1}, \ldots, v_{\ell_r}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_r})$, if $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i, v_j)$ is a partial solution, then there is $\alpha \in \{1, \ldots, r\}$ such that $(v_{\ell_\alpha}, v'_k), (v_{\ell_\alpha}, v''_k) \notin R(C_{\ell_\alpha k})$.

Since s is a solution of the instance I', $(s(x_{\ell_1}), \ldots, s(x_{\ell_r}), s(x_i), s(x_j))$ is necessarily a partial solution, so there is $\alpha \in \{1, \ldots, r\}$ such that we have $(s(x_{\ell_\alpha}), v'_k), (s(x_{\ell_\alpha}), v''_k) \notin R(C_{\ell_\alpha k})$, which implies $(s(x_{\ell_\alpha}), v_k) \notin R(C_{\ell_\alpha k})$. This is a contradiction since s is a solution of the instance I' with $s(x_k) = v_k$.

We can deduce that m-wBTP-merging preserves satisfiability.   □

The BTP-merging rule [4] can be seen as 0-wBTP-merging since it is based on zero support variables. The following proposition establishes the link between the different versions of merging based on BTP.

Proposition 3. *In an n-variable binary CSP, if a pair of values $v'_k, v''_k \in D(x_k)$ satisfies m-wBTP then it satisfies $(m + 1)$-wBTP (for $0 \leq m \leq n - 4$).*

The BTP-merging rule generalises both neighbourhood substitution [12] and virtual interchangeability [15]. As m-wBTP-merging generalises BTP-merging for all $m \geq 0$, the following result follows immediately:

Corollary 1. *m-wBTP-merging generalises neighbourhood substitution and virtual interchangeability.*

Besides the fact that m-wBTP-merging preserves satisfiability, it is also possible to reconstruct in polynomial time all solutions to the original instance I from the solutions from an instance I' obtained by applying a sequence of m-wBTP-mergings. What is more, the reconstruction of a solution to I from a solution to I' can be achieved in time which is linear in the size of I. It suffices to apply the same algorithm as in the case of BTP-merging [4].

4   A Maximal Value-Merging Condition

It is well known that any pair of values which satisfies BTP can be merged while preserving satisfiability [4]. We have shown that a pair of values which does not satisfy BTP can nevertheless be merged while preserving satisfiability

if this pair satisfies m-wBTP. Thus, in an obvious sense, BTP-merging is not a maximal value-merging condition. A value-merging condition is *maximal* if the merging of any other pair of values not respecting the condition necessarily leads to a modification of the satisfiability of some instance. In this section, we show that m-wBTP is a maximal value-merging condition when $m = n - 3$.

Theorem 1. *In an unsatisfiable n-variable binary CSP instance, there is no pair of values not satisfying m-wBTP for $m = n - 3$ and which can be merged while preserving satisfiability.*

Proof. Let I be an unsatisfiable n-variable CSP instance and let $v'_k, v''_k \in D(x_k)$ be a pair of values which does not satisfy m-wBTP for $m = n - 3$. By the definition of m-wBTP-merging, there is a broken triangle (v_i, v_j, v'_k, v''_k), with $v_i \in D(x_i)$ and $v_j \in D(x_j)$, and there is $(v_{\ell_1}, \ldots, v_{\ell_m}) \in D(x_{\ell_1}) \times \ldots \times D(x_{\ell_m})$, where $\{x_{\ell_1}, \ldots, x_{\ell_m}\} = X \setminus \{x_i, x_j, x_k\}$, such that $(v_{\ell_1}, \ldots, v_{\ell_m}, v_i, v_j)$ is a partial solution and for all $\alpha \in \{1, \ldots, m\}$ we have $(v_{\ell_\alpha}, v'_k) \in R(C_{\ell_\alpha k})$ or $(v_{\ell_\alpha}, v''_k) \in R(C_{\ell_\alpha k})$.

We have a broken triangle, and so: $(v_i, v''_k) \notin R(C_{ik})$, $(v_j, v'_k) \notin R(C_{jk})$, $(v_i, v'_k) \in R(C_{ik})$ and $(v_j, v''_k) \in R(C_{jk})$. We also have, for all $\ell \in \{\ell_1, \ldots, \ell_m\}$:

- $(v_\ell, v'_k) \in R(C_{\ell k})$ or
- $(v_\ell, v''_k) \in R(C_{\ell k})$.

After merging, and by definition of merging, the new merged value v_k satisfies $(v_\ell, v_k) \in R(C_{\ell k})$ for all $\ell \in \{\ell_1, \ldots, \ell_m\} \cup \{i, j\}$. We obtain a solution given by $v_{\ell_1}, \ldots, v_{\ell_m}, v_i, v_j$ and v_k. Thus, we have introduced a solution which did not exist in the original instance since $(v_i, v''_k) \notin R(C_{ik})$ and $(v_j, v'_k) \notin R(C_{jk})$. It follows that the merging of any pair of values which does not satisfy m-wBTP does not preserve satisfiability. □

A valid value-merging condition has to guarantee that an unsatisfiable instance does not become satisfiable after merging. We can therefore deduce the following corollary.

Corollary 2. $(n-3)$-*wBTP is a maximal value-merging condition.*

5   wBTP and Variable Elimination

BTP allows value-merging [4], variable elimination [2,3] and the definition of a tractable class [7]. There are several distinct generalisations of BTP according to the desired property. m-wBTP is a generalisation of BTP which allows us to reduce the size of domains via value-merging. m-wBTP is a less restrictive condition than BTP and thus allows more mergings than BTP. On the other hand, this gain in the number of mergings is counterbalanced by the fact that m-wBTP does not allow variable elimination.

In [2], it was shown that, for a given variable x_k of an arc-consistent binary CSP instance I, if there is no broken triangle on any pair of values of $D(x_k)$, then eliminating the variable x_k from I preserves satisfiability. We now show that this is not the case for m-wBTP when $m > 0$.

Proposition 4. *Given a variable x_k of an arc-consistent binary CSP instance I, even if each pair of values in $D(x_k)$ satisfies m-wBTP, where $m \geq 1$, eliminating variable x_k can change the satisfiability of I.*

Proof. Let I be the binary CSP instance defined on four variables $x_1, \ldots, x_4$ with $D(x_i) = \{0, 1, 2\}$ $(i = 1, \ldots, 4)$ and the following constraints: $x_1 = x_2$, $x_2 = x_3$, $x_3 = x_1$, $x_1 = (x_4 + 1) \bmod 3$, $x_2 = (x_4 - 1) \bmod 3$, $x_3 = x_4$. This instance is arc-consistent. There are three partial solutions $(0, 0, 0)$, $(1, 1, 1)$ and $(2, 2, 2)$ on variables x_1, x_2, x_3, but I does not have a solution. Therefore, the elimination of variable x_4 does not preserve the satisfiability of the instance (see Fig. 5).
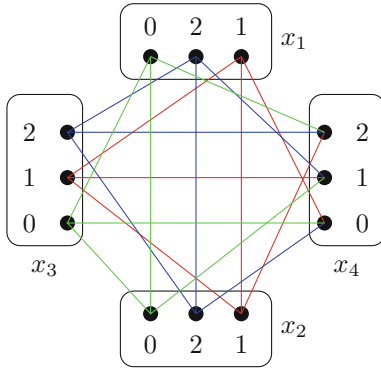
Fig. 5. An unsatisfiable CSP instance in which each pair of values in $D(x_4)$ satisfies 1-wBTP but the elimination of x_4 introduces three solutions.

Let x_i, x_j, x_ℓ be the variables x_1, x_2, x_3 (in any order). There are three broken triangles (v_i, v_j, v'_4, v''_4) on the variables x_i, x_j, x_4 (the weakly broken triangles are represented by three different colours in Fig. 5): in each of these broken triangles, we have $v_i = v_j$. For each of these broken triangles, there is exactly one partial solution of the form (v_ℓ, v_i, v_j) on the variables x_ℓ, x_i, x_j because we necessarily have $v_\ell = v_i = v_j$. By the choice of constraints, the values v_ℓ, v_i, v_j are compatible with three different values in $D(x_4)$. We can deduce that (v_ℓ, v'_4), $(v_\ell, v''_4) \notin R(C_{\ell 4})$ since, by the definition of a broken triangle, each of the values v'_4, v''_4 is compatible with one of the values v_i, v_j. Thus, each pair of values $v'_4, v''_4 \in D(x_4)$ satisfies 1-wBTP.

We have exhibited an instance I such that each pair of values in $D(x_4)$ satisfies 1-wBTP, but eliminating the variable x_4 changes the satisfiability of I. For values of $m > 1$, it suffices to add $m - 1$ other non-constrained variables to the instance I. $\square$

In the instance I in the proof of Proposition 4, each pair of values in the domain $D(x_4)$ satisfies 1-wBTP. However, after having performed the merging of two values, the two remaining values no longer satisfy 1-wBTP and cannot be merged.

6   wBTP and Tractability

In order to compare m-wBTP and other generalisations of BTP defining tractable classes, we extend the definition of m-wBTP in a natural way to instances.

Definition 6. *Given a constant $m \leq n - 3$, a binary CSP instance I with a variable-order $<$ satisfies m-wBTP relative to this order if for all variables x_k, each pair of values in $D(x_k)$ satisfies m-wBTP in the sub-instance of I on variables $x_i \leq x_k$.*

A lighter version of BTP, called k-BTP, which allows certain broken triangles, has recently been defined [8]. Binary CSP instances which satisfy both strong k-consistency and k-BTP constitute a tractable class.

Definition 7 (k-BTP [8]). *A binary CSP instance I satisfies the k-BTP property for a given k ($2 \leq k < n$) relative to a variable order $<$ if, for all subsets of variables $x_{i_1}, x_{i_2}, \ldots, x_{i_{k+1}}$ such that $x_{i_1} < x_{i_2} < \cdots < x_{i_{k+1}}$, there is at least one pair of variables $(x_{i_j}, x_{i_{j'}})$ with $1 \leq j < j' \leq k$ such that there is no broken triangle on x_{k+1} relative to x_{i_j} and $x_{i_{j'}}$.*

Unfortunately, and unlike m-wBTP, the k-BTP property cannot be used for merging values when k is strictly greater than 2 (we recall that 2-BTP = BTP). As an example, the instance of Fig. 6(a) satisfies 3-BTP. To see this, observe that there is no broken triangle on x_k relative to x_i and x_ℓ. But, if we merge v'_k and v''_k, this CSP instance becomes satisfiable whereas it was not initially. Therefore, k-BTP (for k strictly greater than 2) is not a valid value-merging condition. We can also note that k-BTP ($k > 2$) and m-wBTP ($m > 0$) are incomparable, since it can happen that m-wBTP-merging can authorize more broken triangles than k-BTP. For example, the instance in Fig. 6(b) satisfies 1-wBTP but not 3-BTP: there are broken triangles on the variable x_k for each pair of other variables, but in each case the fourth variable is a support variable.

Naanaa has given two other generalisations of BTP which define tractable classes [18,19]. It has been shown [8] that the notion of directional rank $k - 1$ [18] strictly generalises k-BTP. We can deduce that the example of Fig. 6(a) has directional rank 2, which shows that directional rank k (for $k \geq 2$) cannot be used to merge values (knowing that the case $k = 1$ corresponds to BTP).

The notion WBTP [19] inspired our definition of 1-wBTP, but is different. We first give the definition of WBTP before showing that it can be seen as a strictly stronger condition than 1-wBTP (and thus leads to less mergings).

Definition 8 (WBTP [19]). *A binary CSP instance equipped with an order $<$ on its variables satisfies WBTP (Weak Broken Triangle Property) if for each triple of variables $x_i < x_j < x_k$ and for all $v_i \in D(x_i)$, $v_j \in D(x_j)$ such that $(v_i, v_j) \in R(C_{ij})$, there is a variable $x_\ell < x_k$ such that when $v_\ell \in D(x_\ell)$ is compatible with v_i and v_j, then $\forall v_k \in D(x_k)$,*

$$(v_\ell, v_k) \in R(C_{\ell k}) \Rightarrow ((v_i, v_k) \in R(C_{ik}) \ \wedge \ (v_j, v_k) \in R(C_{jk}))$$
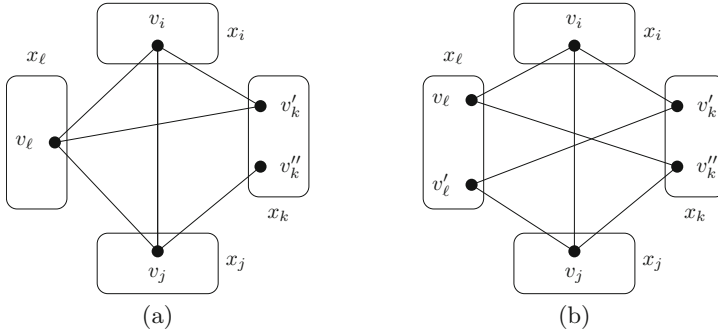
Fig. 6. (a) An instance which does not satisfy 1-wBTP but does satisfy 3-BTP, for the variable ordering $x_\ell < x_i < x_j < x_k$. (b) An instance which satisfies 1-wBTP but does not satisfy 3-BTP, for the variable ordering $x_\ell < x_i < x_j < x_k$.

Proposition 5. *If a binary CSP instance equipped with an order $<$ on its variables satisfies WBTP, then it satisfies 1-wBTP for each pair of values in the domain of the last variable (relative to the order $<$).*

Proof. Suppose that the binary CSP instance I satisfies WBTP for the variable order $<$ and let x_k be the last variable of I according to this order. Suppose, for a contradiction, that I does not satisfy 1-wBTP on a pair of values $v'_k, v''_k \in D(x_k)$. Then, by the definition of 1-wBTP, there is a broken triangle (v_i, v_j, v'_k, v''_k) with $v_i \in D(x_i)$, $v_j \in D(x_j)$, $v'_k, v''_k \in D(x_k)$ such that there is no variable $x_\ell \in X \setminus \{x_i, x_j, x_k\}$ such that $\forall\, v_\ell \in D(x_\ell)$ compatible with v_i and v_j, we have $(v_\ell, v'_k), (v_\ell, v''_k) \notin R(C_{\ell k})$.

But WBTP guarantees the existence of a variable $x_\ell < x_k$ such that $\forall v_\ell \in D(x_\ell)$ compatible with v_i and v_j, we have $\forall v_k \in D(x_k)$,

$$(v_\ell, v_k) \in R(C_{\ell k}) \Rightarrow ((v_i, v_k) \in R(C_{ik}) \,\wedge\, (v_j, v_k) \in R(C_{jk}))$$

The existence of the broken triangle (v_i, v_j, v'_k, v''_k) implies that $x_\ell \notin \{x_i, x_j\}$ and so $x_\ell \in X \setminus \{x_i, x_j, x_k\}$. On the other hand, since (v_i, v_j, v'_k, v''_k) is a broken triangle,

$$(v_i, v_k) \in R(C_{ik}) \wedge (v_j, v_k) \in R(C_{jk})$$

is false for $v_k \in \{v'_k, v''_k\}$. We can deduce that $(v_\ell, v'_k), (v_\ell, v''_k) \notin R(C_{\ell k})$, a contradiction. $\qquad\square$

Imposing WBTP is strictly stronger than imposing 1-wBTP. WBTP imposes a condition on each value $v_k \in D(x_k)$ relative to the *same variable* x_ℓ, whereas 1-wBTP (for each pair of values $v'_k, v''_k \in D(x_k)$) imposes an equivalent condition but for which the variable x_ℓ can vary according to the values v'_k, v''_k. The instance in Fig. 7 satisfies 1-wBTP but not WBTP because:

- only variable x_{ℓ_2} supports (v_i, v_j, v'_k, v''_k),
- only variable x_{ℓ_1} supports (v_i, v_j, v'_k, v'''_k),
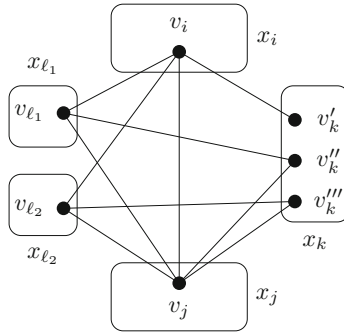
Fig. 7. An instance that satisfies 1-wBTP but not WBTP.

Therefore, there is no variable which supports at the same time the broken triangles (v_i, v_j, v'_k, v''_k) and (v_i, v_j, v'_k, v'''_k).

WBTP defines a tractable class [19]. We now show that this is also true for m-wBTP.

Definition 9. *Let I be a m-wBTP binary CSP instance on variables $x_1, \ldots, x_n$ ordered by $<$.*

- *The **BT-variable set** B_k of x_k is the set of the variables $x_i < x_k$ such that there is a broken triangle on x_k relative to x_i (and some other variable $x_j < x_k$).*
- *A **shield set** S_k of x_k is a set of variables $x_\ell < x_k$ such that for each broken triangle (v_i, v_j, v'_k, v''_k) on x_k relative to variables $x_i, x_j < x_k$, each partial solution $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i, v_j)$ of its support variables, has a shield variable $x_{\ell_\alpha} \in S_k$.*
- *The **BT-width** of x_k is the smallest value of $|B_k \cap S_k|$ among all shield sets S_k of x_k. The BT-width of I is the maximum BT-width of its variables.*

Observe that for constants b and m, it is possible to determine in polynomial time whether a given instance (with a fixed variable order) has BT-width less than or equal to b (by exhaustive search). The BT-width provides an upper bound on the minimum level of consistency required to solve an instance, as demonstrated by the following theorem.

Theorem 2. *If a m-wBTP binary CSP instance I has BT-width b and is strong directional $\max(2, b+1)$-consistent, then I has a solution.*

Proof. Let I be a binary CSP instance which has BT-width b and is directional $(b+1)$-consistent. For simplicity of presentation, we suppose that the variable order is $x_1 < \ldots < x_n$. We suppose that it has a partial solution $\sigma = (v_1, \ldots, v_{k-1})$ on the variables $(x_1, \ldots, x_{k-1})$. We will show that this partial solution can be extended to a partial solution on $(x_1, \ldots, x_k)$. The base case of the induction is easily seen to be true, since by arc consistency there is necessarily a partial solution on the first two variables.

Let B_k be the set of the BT-variables of x_k and let S_k be a shield set of x_k such that $|B_k \cap S_k| \leq b$. By directional $(b+1)$-consistency, any partial solution on the variables $B_k \cap S_k$ can be extended to variable x_k. Therefore $\exists v_k \in D(x_k)$ such that

$$\forall x_i \in B_k \cap S_k, \ (v_i, v_k) \in R(C_{ik}) \tag{1}$$

Denote by $B_k(\sigma)$ the variables $x_i \in B_k$ such that there is a broken triangle of the form (v_i, v_j, v_k', v_k'') on x_k (where v_i, v_j are assignments from σ). Similarly, let $S_k(\sigma)$ be the variables of S_k which shield such broken triangles. Let $N_k(\sigma)$ be the variables $x_i < x_k$ such that $x_i \notin B_k(\sigma)$. The sub-instance of I on variables $N_k(\sigma) \cup \{x_k\}$ has no broken triangles on x_k. Therefore, $\exists u_k \in D(x_k)$ such that $(v_i, u_k) \in R(C_{ik})$ for all $x_i \in N_k(\sigma)$ [7]. If $N_k(\sigma) = \emptyset$, then u_k is simply an arbitrary element of $D(x_k)$. We will show that one of $(v_1, \ldots, v_{k-1}, v_k)$ or $(v_1, \ldots, v_{k-1}, u_k)$ is a partial solution.

Suppose, for a contradiction, that this is not the case. Then $\exists x_i, x_j < x_k$ such that $(v_i, u_k) \notin R(C_{ik})$ and $(v_j, v_k) \notin R(C_{jk})$. We must have $x_i \in B_k(\sigma)$ and $x_j \notin B_k \cap S_k$. Since $x_i \in B_k(\sigma)$, there is a broken triangle (v_i, v_h, v_k', v_k'') on x_k with $(v_i, v_k') \in R(C_{ik})$. This broken triangle must have a shield variable $x_\ell \in S_k(\sigma)$. If $x_\ell \in N_k(\sigma)$, then $(v_\ell, u_k) \in R(C_{\ell k})$. We also have $(v_\ell, v_i) \in R(C_{ik})$ (by the definition of a partial solution) and $(v_\ell, v_k') \notin R(C_{\ell k})$ (by definition of a support variable). Since, by assumption, $(v_i, u_k) \notin R(C_{ik})$, we have a broken triangle (v_i, v_ℓ, v_k', u_k) which is impossible since $x_\ell \in N_k(\sigma)$ and hence cannot participate in such a broken triangle. So the shield variable x_ℓ belongs to $B_k(\sigma) \cap S_k(\sigma)$. By (1), we have $(v_\ell, v_k) \in R(C_{\ell k})$. Suppose now that $(v_i, v_k) \notin R(C_{ik})$. Then we have a broken triangle (v_i, v_ℓ, v_k', v_k). This broken triangle must have a shield variable x_m. By the same argument as for x_ℓ, we can deduce that $x_m \in B_k(\sigma) \cap S_k(\sigma)$. However, this contradicts (1) since we have $(v_m, v_k) \notin R(C_{mk})$ (since x_m is a shield variable of the broken triangle (v_i, v_ℓ, v_k', v_k)). It follows that $(v_i, v_k) \in R(C_{ik})$. Indeed, we have shown

$$\forall x_i \in B_k(\sigma), \ (v_i, u_k) \notin R(C_{ik}) \ \Rightarrow \ (v_i, v_k) \in R(C_{ik}) \tag{2}$$

Now, if $x_j \in N_k(\sigma)$ we have a broken triangle (v_i, v_j, v_k, u_k), which is in contradiction with the definition of $N_k(\sigma)$, so we must have $x_j \in B_k(\sigma)$. Now, by (2) and our assumption that $(v_j, v_k) \notin R(C_{jk})$, we can deduce that $(v_j, u_k) \in R(C_{jk})$. We then have a broken triangle (v_i, v_j, v_k, u_k). This broken triangle must have a shield variable x_p. By definition of a shield variable, we must have $(v_p, v_k), (v_p, u_k) \notin R(C_{pk})$. But this is impossible since $x_p \in N_k(\sigma) \Rightarrow (v_p, u_k) \in R(C_{pk})$ and, by (2), $x_p \in B_k(\sigma) \Rightarrow (v_p, u_k) \in R(C_{pk}) \lor (v_p, v_k) \in R(C_{pk})$.

This contradiction shows that one of $(v_1, \ldots, v_{k-1}, v_k)$ or $(v_1, \ldots, v_{k-1}, u_k)$ is a partial solution. By induction, I has a solution. $\square$

Naanaa showed that a binary arc-consistent CSP instance satisfying WBTP always has a solution [19]. We can observe that this is a special case of Theorem 2 since a WBTP instance has BT-width of 1.

An open question is whether it is possible to determine, in polynomial time, the existence of some variable order for which a given instance has BT-width b even for $b = 1$.

7 Experimental Results

In order to test the applicability of our merging rules, and in particular 1-wBTP-merging, we carried out an experimental study on all the binary benchmark instances of the 2008 international CSP solver competition[1] (a total of 3,795 instances). The 1-wBTP-merging algorithm is similar to the algorithm for BTP-merging [5]. More specifically, given a variable x_k, we check for each pair of values $v'_k, v''_k \in D(x_k)$ if these two values are mergeable by 1-wBTP-merging. Once a broken triangle on v'_k, v''_k is found, we search over the other $n - 3$ variables to see if there exists a variable x_ℓ which supports this broken triangle. If we find one, we continue the search for other broken triangles; if not, the test is finished for these two values. Finally, if there are no broken triangles or only weakly broken triangles on the pair v'_k, v''_k, we merge them. We do not attempt to maximize the number of merges since we know that this is an NP-hard problem, even in the case of BTP-merging [4]. We implemented the two merging algorithms to be tested (BTP-merging and 1-wBTP-merging) in C++ within our own CSP library. The experiments were performed on 8 Dell PowerEdge M820 blade servers with two processors (Intel Xeon E5-2609 v2 2.5 GHz and 32 GB of memory) under Linux Ubuntu 14.04.
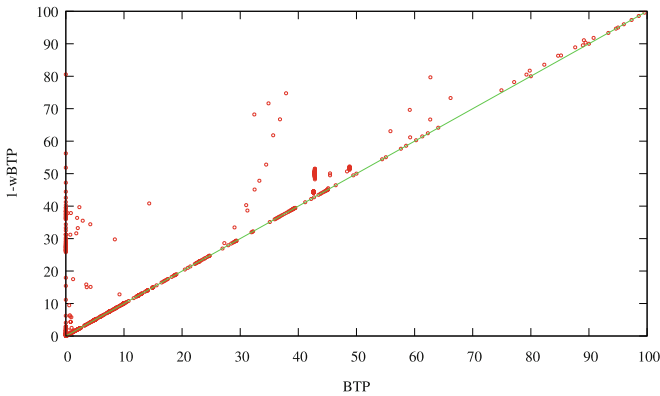
Fig. 8. Comparisons of the percentages of values merged by BTP and 1-wBTP.

For each benchmark instance, we performed BTP-merging and 1-wBTP-merging until convergence with a timeout of one hour. In all, we obtained results for 2,535 out of the 3,795 benchmarks and we succeeded in merging at least one

---

[1] See http://www.cril.univ-artois.fr/CPAI08 for more details.

Table 1. Experimental results on benchmarks.

Family	#benchmarks	#values	BTP-merging	1-wBTP-merging
`BH-4-4`	10	674	322	348
`BH-4-7`	20	2 102	883	932
`ehi-85`	98	2 079	891	1 045
`ehi-90`	100	2 205	945	1 100
`graph-coloring/school`	8	4 473	104	104
`graph-coloring/sgb/book`	26	1 887	534	534
`os-taillard-4`	30	2 932	1 820	1 978
`rlfapScens`	1	8 004	341	1 211
`rlfapScensMod`	6	8 788	2 415	5 169
`subs`	9	1 479	40	517
`langford-2`	22	879	0	233
`langford-3`	20	1 490	0	554
`langford-4`	16	1 784	0	504
`queenAttacking`	7	2 196	0	36

pair of values for 1,001 of these instances. In Table 1, the column #benchmarks shows the number of benchmark instances for which the test finished within the one-hour timeout. The column #values indicates the average total number of values in these benchmarks. The columns BTP-merging and 1-wBTP-merging give the number of merges performed respectively by BTP-merging and 1-wBTP-merging. In Fig. 8, we compare the percentages of domain reduction by BTP-merging and 1-wBTP-merging instance by instance. If, for the majority of instances, the results are comparable, we can observe that for certain instances, 1-wBTP merges significatively more values than BTP. This is notably the case for the instances in the `langford-*` family for which 1-wBTP merges from 25 to 80 % of the values whereas BTP does not merge any.

8   Conclusion

In this paper we have studied value-merging conditions in binary CSP instances, based on a generalisation of BTP. We proposed a family of definitions based on the notion of a weakly broken triangle, which is a broken triangle supported by one or more variables in order to preserve satisfiability after merging.

We have shown that m-wBTP together with different levels of consistency defines a family of tractable classes. Possible links with bounded treewidth are worth investigating. From a practical point of view, it would be interesting to investigate the influence of the order in which merges are performed on the total number of merges. We know that finding the best order in which to perform m-wBTP-merging operations is NP-hard even in the case $m = 0$ [4].

References

1. Carbonnel, C., Cooper, M.C.: Tractability in constraint satisfaction problems: a survey. Constraints **21**(2), 115–144 (2016)
2. Cohen, D.A., Cooper, M.C., Escamocher, G., Živný, S.: Variable elimination in binary CSP via forbidden patterns. In: Proceedings of IJCAI (2013)
3. Cohen, D.A., Cooper, M.C., Escamocher, G., Živný, S.: Variable and value elimination in binary constraint satisfaction via forbidden patterns. J. Comput. Syst. Sci. **81**(7), 1127–1143 (2015)
4. Cooper, M.C., Duchein, A., El Mouelhi, A., Escamocher, G., Terrioux, C., Zanuttini, B.: Broken triangles: from value merging to a tractable class of general-arity constraint satisfaction problems. Artif. Intell. **234**, 196–218 (2016)
5. Cooper, M.C., El Mouelhi, A., Terrioux, C., Zanuttini, B.: On broken triangles. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 9–24. Springer, Heidelberg (2014)
6. Cooper, M.C., Jeavons, P., Salamon, A.: Hybrid tractable CSPs which generalize tree structure. In: Proceedings of ECAI, pp. 530–534 (2008)
7. Cooper, M.C., Jeavons, P., Salamon, A.: Generalizing constraint satisfaction on trees: hybrid tractability and variable elimination. Artif. Intell. **174**, 570–584 (2010)
8. Cooper, M.C., Jégou, P., Terrioux, C.: A microstructure-based family of tractable classes for CSPs. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 74–88. Springer, Heidelberg (2015)
9. Cooper, M.C., Živný, S.: The power of arc consistency for CSPs defined by partially-ordered forbidden patterns. In: Proceedings of LICS (2016)
10. El Mouelhi, A., Jégou, P., Terrioux, C.: Hidden tractable classes: from theory to practice. In: Proceedings of ICTAI, pp. 437–445 (2014)
11. El Mouelhi, A., Jégou, P., Terrioux, C.: A hybrid tractable class for non-binary CSPs. Constraints **20**(4), 383–413 (2015)
12. Freuder, E.C.: Eliminating interchangeable values in constraint satisfaction problems. In: Proceedings of AAAI, pp. 227–233 (1991)
13. Jégou, P.: Decomposition of domains based on the micro-structure of finite constraint satisfaction problems. In: Proceedings of AAAI, pp. 731–736 (1993)
14. Jégou, P., Terrioux, C.: The extendable-triple property: a new CSP tractable class beyond BTP. In: Proceedings of AAAI, pp. 3746–3754 (2015)
15. Likitvivatanavong, C., Yap, R.H.C.: Many-to-many interchangeable sets of values in CSPs. In Proceedings of SAC, pp. 86–91 (2013)
16. Mackworth, A.K.: Consistency in networks of relations. Artif. Intell. **8**, 99–118 (1977)
17. Montanari, U.: Networks of constraints: fundamental properties and applications to picture processing. Artif. Intell. **7**, 95–132 (1974)
18. Naanaa, W.: Unifying and extending hybrid tractable classes of CSPs. J. Exp. Theor. Artif. Intell. **25**(4), 407–424 (2013)
19. Naanaa, W.: Extending the broken triangle property tractable class of binary CSPs. In: Proceedings of the 9th Hellenic Conference on Artificial Intelligence, SETN, pp. 3:1–3:6 (2016)
20. Sabin, D., Freuder, E.C.: Contradicting conventional wisdom in constraint satisfaction. In: Proceedings of ECAI, pp. 125–129 (1994)

A Bounded Path Propagator on Directed Graphs

Diego de Uña[1(✉)], Graeme Gange[1], Peter Schachte[1], and Peter J. Stuckey[1,2]

[1] Department of Computing and Information Systems,
The University of Melbourne, Melbourne, Australia
`d.deunagomez@student.unimelb.edu.au`,
`{gkgange,schachte,pstuckey}@unimelb.edu.au`
[2] Data 61, CSIRO, Melbourne, Australia

Abstract. Path finding is an ubiquitous problem in optimization and graphs in general, for which fast algorithms exist. Yet, in many cases side constraints make these well known algorithms inapplicable. In this paper we study constraints to find shortest paths on a weighted directed graph with arbitrary side constraints. We use the conjunction of two directed tree constraints to model the path, and a bounded path propagator to take into account the weights of the arcs. We show how to implement these constraints with explanations so that we can make use of powerful constraint programming solving techniques using learning. We give experiments to show how the resulting propagators substantially accelerate the solving of complex path problems on directed graphs.

1 Introduction

Path-finding is an important task in (directed) networks. It arises in tasks such as graph layout [7], metabolic networks [25] or collaborative path-finding in video-games [22] among other examples. In many cases, though, side constraints make these problems highly combinatorial and no efficient algorithms exist.

In this paper, we focus on path-finding with distances. In order to do so, we go through preliminary steps to build two propagators from which we build a path propagator that works on the topology of the graph. Then, on top of that, we construct a propagator that takes into account the weights of the arcs to propagate distances.

Given a fixed directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we enforce properties of a graph variable $G = (V, E)$ subgraph of $\mathcal{G}$ using the following constraints:

- *dreachability*$(G, r, \mathcal{G})$ requires all nodes in G are reachable from root r;
- *dtree*$(G, r, \mathcal{G})$ requires that G forms a tree rooted at r;
- *path*$(G, s, d, \mathcal{G})$ ensures that G is a simple path from s to d;
- *bounded_path*$(G, s, d, \mathcal{G}, w, K)$ ensures that G is a simple path from s to d of length no more than K given the weight function w over the arcs of $\mathcal{G}$;

The focus of the present paper is the *bounded_path* propagator. We present two novel explanations for already existing propagation rules. Furthermore, we introduce a new stronger propagation technique with explanations as well. The

explanations for this propagator and its new version are the main contributions of this paper.

Section 2 describes previous work as well as use cases for these propagators. Section 3 gives the necessary technical background to the reader as well as all the graph propagators for unweighted graphs. Section 4 describes *bounded_path*. Section 5 shows an extensive series of experiments justifying the use of these propagators and their explanations.

2   Related Work

The three first constraints announced in the introduction were first introduced as part of CP(Graph) [6] in 2005, using a decomposition approach.

Later, Quesada *et al.* [17] implemented the first reachability propagator and used it as a path constraint in their paper. They make use of simple propagation rules based on depth-first traversals of the graph and on the use of dominator nodes (i.e. nodes that appear in all paths). Nonetheless the asymptotic complexity of their algorithms is substantially greater than ours or those of Fages *et al.* [9] since they use a brute-force algorithm to compute dominator nodes.

Constraints for trees and forests were introduced in [1,2,9]. Although focused on forests, their work used better algorithms improving the work of Quesada *et al.* in [17] to make each individual tree connected. For explanations on the *dtree* constraint, we use the same algorithm as we previously introduced in [5] for undirected graphs.

In order to be self-contained, we describe *dreachability* and *dtree* in the preliminaries section. The propagations are based on previous work presented in [1,9,17]. The explanations are novel although the algorithms are similar to the undirected version which we already introduced [5].

Finding a simple path (no node repetitions) is a classic graph problem with wide applicability. The usefulness of the constraint arises when there are interesting side constraints. Our *path* propagator is based on the Ph.D. thesis by J.-G. Fages [8], which showed how to model the path constraint as a conjunction of *dtree* constraints:

$$path(G, s, d, \mathcal{G}) \Leftrightarrow dtree(G, s, \mathcal{G}) \wedge dtree(G, d, \mathcal{G}^{-1}) \qquad (1)$$

This states that a path from s to d is the intersection of a subtree of $\mathcal{G}$ rooted at s and a tree in $\mathcal{G}^{-1}$ (the graph $\mathcal{G}$ with arcs reversed) rooted at d.

There exist other approaches to finding paths by using *circuit* style propagators [10]. We compare for the first time the tree-based and the circuit-based approaches where both use explanations.

Path finding with distances is one of the most well-studied graph problems, for which very well known fast algorithms exist. Many specific algorithms that handle some form of side constraint are also known. For instance, paths with resource constraints have been very well studied for electrical cars [23] and for bike routes [24]. Another application is the Generalized Shortest Path queries [18, 19] where a person needs to do a series of tasks during their journey and choose

among different places to do them. The *bounded_path* constraint allows us to specify shorter path problems with arbitrary side constraints. It was introduced by Sellman [20,21] with some propagation rules. Our work improves on this.

3   Preliminaries

3.1   Directed Graphs

A *directed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of nodes $\mathcal{V}$ and arcs $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, where $e = (u, v)$ is an arc from $u = tail(e)$ to $v = head(e)$ (drawn '$u \longrightarrow v$', from tail to head). Given arc $e = (u, v)$ its *reverse* arc is $e^{-1} = (v, u)$. The *inverse* $\mathcal{G}^{-1}$ of a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is $(\mathcal{V}, \{e^{-1} \mid e \in \mathcal{E}\})$. A *weighted directed graph* is a graph $\mathcal{G}$ with a weight function $w : \mathcal{E} \to \mathbb{N}^0$ mapping arcs to non-negative *weights*.

3.2   Lazy Clause Generation

Briefly, Lazy Clause Generation (LCG, [16]) is a technique by which CP solvers can *learn* from their mistakes. Propagators are extended to explain their propagations, and the failures they detect. These *explanations* are captured in clauses. When failure is detected, explanations are used to generate concise no-goods that explain why the failure occurred, and these are stored in the solver, preventing the same failure from occurring again. Using SAT technology to access and process explanations and no-goods allows very efficient handling of no-goods, and the reduction in search space for using explanation is usually substantial.

A critical consideration when constructing propagators for an LCG solver are the algorithms to generate concise and precise explanations of the propagation. Naive explanations may end up creating no-goods that are not reusable, while highly complex minimal explanations may require much more computation effort than propagation, and end up slowing down the solver.

3.3   Graph Propagators with Explanations

In order to model a graph variable $G = (V, E)$ subgraph of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in an LCG solver, we use Boolean variables c_n representing whether node $n \in \mathcal{V}$ is chosen to be in V and similarly Boolean variables c_e for each $e \in \mathcal{E}$ representing whether $e \in E$. Eventually the solution is the subgraph $G = (\{n | n \in \mathcal{V} \wedge c_n\}, \{e | e \in \mathcal{E} \wedge c_e\})$.

As we are searching for G, the variables c_n and c_e will become fixed by search or propagation. The propagators we describe here must infer new information as a consequence of the constraint they implement, hence reducing the future search. We say that an arc e is *mandatory* if at the current stage of the search c_e is *true* (and we draw it as '——' in the following figures), *forbidden* ('·····') if c_e is *false* and *unknown* ('- - -') for an unassigned arc. Similarly, we use the same terms for nodes: *mandatory* ('●'), *forbidden* ('⋰') and *unknown* ('○'). Nodes or arcs that are mandatory or unknown are called *available*.

Basic Graph Propagation. We assume that the graph variable G propagates basic graph properties: the endnodes of an arc in the graph are also in the graph. Explanations for this are given in previous work by the authors [5].

Reachability Propagation. The *dreachability* constraint guarantees that all nodes in the subgraph G are reachable from a given node r. Quesada *et al.* [17] first proposed this propagator, although our algorithm is substantially improved by making use of the Lengauer-Tarjan algorithm to find dominators in a digraph [13]. Fages *et al.* [9] already used this algorithm.

Detecting and explaining failure: In order to detect that the current assignment of arcs and nodes in G is invalid, we need to check if all the nodes in G (i.e. mandatory) are reachable from the given node r. We first perform a depth-first search (DFS) in $(\{n \mid c_n \neq false\}, \{e \mid c_e \neq false\})$ starting at r, saving all the nodes visited in a set R. If some mandatory node f is not in R, we need to fail.

To explain why the mandatory node f is not reachable, we need to find forbidden arcs that would have let it be in R if they were not forbidden, similarly to the work in [5] for undirected graphs.

To find these arcs, we perform a DFS in $\mathcal{G}^{-1}$ starting at f, this time following all (reversed) arcs, regardless of their current state. Whenever the head of a forbidden reversed arc $e^{-1} = (t, h)$ is in R, e could have been used to extend the reachable area further and eventually reach f. Therefore, e must be in the explanation (we do not cross e^{-1} in this DFS). We add such arcs to a set F_f. Then an explanation for failure is: $c_r \wedge c_f \wedge \bigwedge_{e \in F_f} \neg c_e \Rightarrow false$. This exact same rule can be applied for propagation to eliminate unreachable nodes.

Finding dominators: During the search, we also make inferences that will accelerate the search. We say that a node t is *dominated* by a node d from r if all paths from r to t go through d. The *immediate dominator* of a node is the dominator that is its closest ancestor. For reachability, immediate dominators of mandatory nodes must be mandatory, otherwise some node (namely t) would not be reachable from r.

Finding immediate dominators in a graph can be done using the Lengauer-Tarjan algorithm [13] in $\mathcal{O}(|\mathcal{E}|\alpha(|\mathcal{E}|, |\mathcal{N}|))$ where α is the inverse Ackerman function. Their algorithm builds an array representation of a so-called *dominator tree* where the parent of a node is its immediate dominator. For our purposes, we apply the algorithm to $(\{n \mid c_n \neq false\}, \{e \mid c_e \neq false\})$.

We assume that the reachability has been ensured and thus all mandatory nodes are reachable from r. To enforce dominators to be in G, we build a queue containing all the mandatory nodes and iterate through the queue until it is empty while making their immediate dominators mandatory (if they are not already) and enqueueing them. This way, all the nodes in the path between r and some mandatory node t in the dominator tree become mandatory.

Now, we need to explain why each dominator that we fix is mandatory. Explaining this inference comes down to explaining the failure that would happen

if the dominator d of t was forbidden. We compute a partition of nodes P from which t can be visited without going through the dominator by performing a DFS starting at t in $(\{\mathcal{V}\backslash\{d\}, \{e^{-1} \mid c_e \neq false\})$. Now we find alternative ways to get to any mandatory node beyond the dominator (that is, not in P) without using d. For that, we perform another DFS in $(\{\mathcal{V}\backslash\{d\}, \{e^{-1} \mid e \in \mathcal{E}\})$ starting in t, this time allowing the use of forbidden arcs. Whenever a forbidden reversed arc $e^{-1} = (t, h)$ has its tail on the set P but its head is a node outside of P we know that e would have allowed us to short-circuit d if it was not forbidden. Let F_t be the set of such arcs. The explanation for making a dominator mandatory is: $c_r \wedge c_t \wedge \bigwedge_{e \in F_t} \neg c_e \Rightarrow c_d$.

Finding bridges: Additionally, if any mandatory node n (other than r) has only one incoming arc that is not forbidden, that arc can be set as mandatory (if it is not already). This is because lacking that arc would make that node unreachable. That arc is called a bridge. The explanation for including this arc e in G is trivial: $c_n \wedge \bigwedge_{e_i \in to(n)\backslash\{e\}} \neg c_{e_i} \Rightarrow c_e$, where $to(n)$ is the set of arcs incident to n. Similarly, if n no longer has any incident arcs available, we have to set it to $false$, or fail if it is mandatory, explained by $\bigwedge_{e_i \in to(n)} \neg c_{e_i} \Rightarrow \neg c_n$.

Directed Tree Propagator. Trees are connected graphs, therefore *dtree* inherits from *dreachability*. Additionally, trees cannot contain any cycle (whether it is directed or not). Maintaining this condition is the task of *dtree*.

For this propagator, the algorithm is trivial. The use of the adequate data structure to detect cycles is what makes the whole propagator. We use the Rerooted Union-Find (RUF) data structure [5] to detect cycles and retrieve explanations. This yields a propagator identical to the one for undirected graphs [5] since cycle detection in undirected and directed graphs is equivalent as far as trees are concerned.

4   Bounded Path Propagator

As we will see in the experiments, the decomposition of the path constraint as two trees (Eq. 1) is not competitive for solving shortest path problems when compared to the alternative circuit formulation by Francis *et al.* [10]. For this reason, we needed a bounded path propagator to enhance optimality proving.

In this section we present a $bounded\_path(P, s, d, \mathcal{G}, w, K)$ propagator that ensures that the weight of the simple path P from s to d in $\mathcal{G}$ is no more than K. The weights of the arcs are given by the function $w : \mathcal{E} \mapsto \mathbb{N}^0$. The propagations in Sects. 4.1 and 4.2 were already introduced by Sellman [20,21], without explanation. As we will see in the experimental section, the explanations greatly improve these propagations.

4.1   Propagating Simple Distances

This constraint fails when there is no path from s to d of cost no more than K. This property naturally extends to all nodes in the path: the distance from s to

any node n in P must be no more than K. The best correct lower bound for this is obviously the shortest path from s to $n \in P$: if the shortest path is longer than K, then no solution of cost less than or equal to K exists.

We compute the shortest path from s to every node in $(\mathcal{V}, \{e \mid c_e \neq \textit{false}\})$ (i.e. avoiding forbidden arcs) using Dijkstra's algorithm. This yields the shortest *available* path from s to all nodes. If the cost of the path to a node n is greater than K, we can forbid n. This reasoning can be applied in both directions: d cannot be further than K from any node n in the path. For this reason, we also apply this rule starting Dijkstra's algorithm at d on the reversed graph.

To explain this inference we need to find (at least a superset of) the arcs that made the path to n too expensive. Let F_n be that set of arcs (initially empty). Let δ_x be the shortest available path from s to some node x, and δ_x^{-1} the shortest available path from x to d. Any arc $e = (u, v)$ such that $\delta_u + w[e] + \delta_v^{-1} \leq K$ can be used to connect s to d in no more than K (we say e is in a *short-enough* path). We can easily keep track of those arcs since both runs of Dijkstra's algorithm yield δ_u and δ_v^{-1}. When such an arc is forbidden, a feasible path is removed from the graph. We then add e to F_n. Eventually, F_n contains *all* the arcs causing n to be too far from either s or d. We have the following Theorem:

Theorem 1. *Let $[\![\delta_d \leq K]\!]$ be the literal stating that δ_d (i.e. the length of P) should be less than or equal to K (K is typically a variable). Then, $[\![\delta_d \leq K]\!] \wedge \bigwedge_{e_f \in F_n} \neg c_{e_f} \Rightarrow \neg c_n$ is a valid explanation for why n cannot be in G.*

Note the explanation set F_n is the same for any node n further than K from the source, its not specific to a particular n. We will address this flaw and give an example in Fig. 2 later on. The explanations can be used to explain failure too.

These explanations can be computed very efficiently by storing a function giving constant time access to whether an arc has been in a short-enough path. Upon removal of an arc e, we add it to F_n if e has been in a short-enough path.

4.2   Propagating Combined Distances

The previous rule removes any node that is too far from the source or too far from the destination to be in the path P, or detects failure. In addition, we can consider nodes through which a path from s to d would be longer than K and filter them. Similarly we can filter arcs.

Proposition 1. *Let δ_u be the cost of the shortest path from s to u, and let δ_u^{-1} be the cost of the shortest path from u to d. If $\delta_u + \delta_u^{-1} > K$, then u cannot be in the path from s to d of cost less than or equal to K.*

Proposition 2. *Let $e = (u, v)$ be an arc of cost $w[e]$. Let δ_u and δ_v^{-1} be the cost of the shortest paths from s to u and v to d respectively. If $\delta_u + w[e] + \delta_v^{-1} > K$, then e cannot be in a path from s to d of cost less than or equal to K.*

We use these observations to filter out nodes and arcs that cannot participate in the path from s to d.

Algorithm 1. Shortest path from s to d containing all mandatory nodes M.

```
 1: procedure DPBOUND(𝒢, s, d, ns = {c_n|n ∈ 𝒱}, es = {c_e|e ∈ ℰ}, w, M)
 2:     Q ← new PriorityQueue(); Q.push((s, {s}, 0));
 3:     tables[s][{s}] ← 0                                    ▷ One table per node
 4:     while ¬Q.empty() do
 5:         (u, m_p, γ) ← Q.top(); Q.pop()
 6:         if tables[u].contains(m_p) ∧ tables[u][m_p] < γ then continue;
 7:         for all e = (u, v) ∈ {e|e ∈ ℰ} do
 8:             if c_e = false then continue;
 9:             if ¬tables[v].contains(m_p) ∨ (tables[v][m_p] > γ + w[e]) then
10:                 tables[v][m_p] ← γ + w[e]
11:                 Q.push(v, m_p ⊕ v, γ + w[e])   ▷ S ⊕ v adds v to set S iff v ∈ M
12:     return tables[d][M]
```

To explain these propagations, we note that if the filtered element (either node or arc) was mandatory, we would have to fail. Thus the explanations are the same as given in Theorem 1 (applied to the node or the arc we are propagating here). These explanations can be used for failure if either u or e is mandatory.

4.3   Stronger Bounding Using Dynamic Programming

Although the implementation of *bounded_path* explained above proves to be useful, the bound is too weak if there are many intermediate nodes. For this reason, we developed a dynamic programming (DP) lower bound. If the previous one does not prune, we run a more expensive DP algorithm to find the shortest path from s to d containing *all* the mandatory nodes.

The algorithm is similar to Dijkstra's, but our priority queue stores more information. Each entry is a tuple (u, m_p, γ): a node u, the set of mandatory nodes m_p visited in some path p leading to u and the cost of p. As usual, the priority is on the cost.

We associate a hash-table to each node n that maps sets of mandatory nodes (encoded as bit-sets in our implementation) to the cost of visiting those nodes before reaching n. Formally the tables are functions $tables[n] : (M' \subseteq M) \mapsto \mathbb{N}$.

Then, when a tuple (u, m_p, γ) is retrieved from the queue, the algorithm considers each available arc (u, v). For each neighbor node v, we first check if there is a set m'_p in its table such that $m'_p = m_p$. If m'_p exists and its associated cost is greater than $\gamma + w[(u, v)]$, we update the entry on v's table, and enqueue $(v, m_p \oplus v, \gamma + w[(u, v)])$ (where $\oplus$ adds v to m_p iff v is mandatory, and returns m_p otherwise). If such m'_p does not exist, we add that same entry to v's table and enqueue it. We do not need to enqueue or update any table if m'_p exists and its associated cost is less than $\gamma + w[(u, v)]$. The cost of the shortest path to d containing all the nodes will be found in d's table. If such path does not exist, we simply return an error code and fail with the naive explanation (all the fixed arcs and nodes). In practice, this rarely happens.

Notice that this algorithm does not give simple paths, and therefore it does not give an exact lower bound. Indeed, if we did, we would need to keep track of all the states in the path, making the state space grow too quickly. Instead we only keep track of the mandatory nodes visited.

The explanation for pruning is the same as in Theorem 1, but we need to add the conjunction of c_n for all the mandatory nodes $n \in M$. Note that the asymptotic complexity of this algorithm is $\mathcal{O}(n2^{|M|}|M|log(n))$, hence the state may grow prohibitively. We will study solutions to this issue in the next subsection. Nonetheless, as we will see in the results, this explosion rarely happens since the higher the number of mandatory nodes, the smaller the choice in arcs.

Limiting State Explosion in the DP Propagation

Strongly Connected Components: Some basic inference we can take into account to reduce the state explosion is based on strongly connected components (SCC) of the current graph. There is no point for the DP algorithm to take an arc leaving SCC A if it has not yet visited all the mandatory nodes in A.

We use Kosaraju's algorithm [12] to compute SCCs. We then label the SCCs as follows. Let m be the number of SCCs containing at least one mandatory node (we call them mandatory SCCs). The SCC D containing the destination node d is labeled m. All other mandatory SCCs are numbered with the number of the lowest numbered SCC they can reach minus one. All non-mandatory SCCs are numbered with the lowest numbered SCC they can reach. It is easy to do this in linear time using a topological sort on the graph of SCCs. We call this *levels* and we denote the level of an SCC A by $l(A)$. Then, if an arc e goes from A to B such that $l(B) > l(A)+1$, by crossing it we would skip some mandatory SCC to which we can never go back. Similarly, if A is mandatory and $l(B) = l(A)+1$ we only cross e if we have visited all mandatory nodes of A, otherwise we would not be able to get back to A to finish visiting the mandatory nodes in A.

This process can greatly accelerate the DP algorithm without losing pruning power. Nonetheless, because during the search the partially assigned graphs tend to have a succession of SCCs of only one node followed by a big SCC containing all the unassigned nodes, we often did not see a benefit from this. It is, however, very worthwhile running at the root level. As a simple example, consider the graph in Fig. 1: it takes 0.03 s to solve the problem using the SCC labeling, but 22.72 s without it (same number of nodes and conflicts).

Each node on the left graph is an SCC of the form of the right graph (from [17]). Each SCC contains 3 random mandatory nodes. All edges have weight 1.
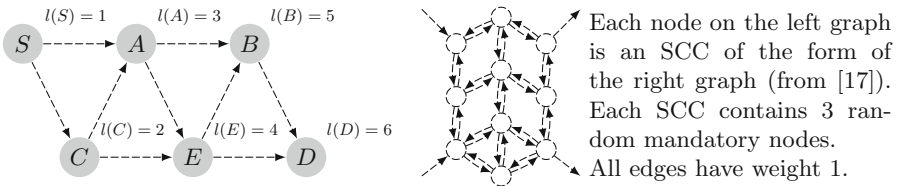
Fig. 1. Example of use of SCCs to accelerate Algorithm 1.

Clustering mandatory nodes: Further acceleration can be achieved by reducing the number of mandatory nodes to visit. To decide which ones to ignore, we use the k-means clustering algorithm [11] on the set M of mandatory nodes. We use the centroids of the clusters only as mandatory nodes (i.e., we have as many mandatory nodes as clusters, treating the non-centroid nodes as unknown). Because the centroids tend to be equidistant to the other nodes in the cluster, the DP tends to also use some of the other mandatory nodes, thus visiting more than just the centroids. Also, since k-means has some inherent randomness, we have different clusters every time, which is also beneficial for the lower bound.

This has huge performance effects, but is a double-edged sword: the DP gets faster but we prune much less often as the bound is not as high. In order to regulate this, we use a simple heuristic based on the time spent by the DP. If the DP algorithm with C clusters takes less than x seconds, we increase C by 1, if it takes more than y seconds, we lower it. For the experiments where we used clustering, we chose $x = 0.5\,\mathrm{s}$, $y = 8.0\,\mathrm{s}$ and started with $C = 5$.

4.4   Improving the Explanations

So far, the explanations for *bounded_path* have been the set of forbidden arcs that were in a short-enough path at some point (see Sect. 4.1). One problem with these explanations is that they are not targeted. It is easy to see that some of the arcs in the explanations may have nothing to do with the fact that some specific node n is too far from the source. We now provide better explanations.

Simple and Combined Distances Propagation. First, during the propagation we use Dijkstra's algorithm on the available graph. This leaves a label on each node indicating how far it is from the source. These labels are noted $\delta_n, \forall n \in V$. Nodes not visited have label $\delta_n = \infty$.

Let n be a node that is at distance δ_n more than the limit K from the source. Algorithm 2 returns a set of forbidden arcs that explain why $\delta_n > K$.

Algorithm 2. Explaining why n is at distance more than K from the source.

1: **procedure** EXPLAINDIST($\mathcal{G}, s, n, \{\delta_u | u \in \mathcal{V}\}, K$)   ▷ We consider all arcs in $\mathcal{G}$
2:     $Q \leftarrow new\, PriorityQueue()$; $Q.push((n, 0))$; $X = \emptyset$; $cost = [\infty | v \in \mathcal{V}]$
3:     **while** $\neg Q.empty()$ **do**
4:         $(u, \delta_u^{-1}) \leftarrow Q.top()$; $Q.pop()$ ▷ δ_u^{-1} : cost of the shortest path from n to u
5:         **if** $u = s$ **then break**   ▷ Reached start s
6:         **for all** $e = (v, u) \in \{e | e \in \mathcal{E}\}$ **do**   ▷ Notice that we take arcs backwards
7:             **if** $e \in F \wedge \delta_v + w[e] + \delta_u^{-1} \leq K$ **then** $X = X \cup \{\, e \,\}$
8:             **else if** $cost[v] > \delta_u^{-1} + w[e]$ **then**
9:                 $cost[v] = \delta_u^{-1} + w[e]$ ▷ Update cost
10:                $Q.push(v, cost[v])$ ▷ Overwrites previous instances of v in Q
11:     **return** X
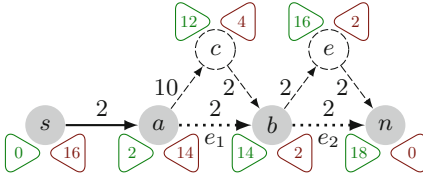
K = 10, $\delta_n = 18 > K$, thus we fail.
Algorithm 2, starting at n (line 7):
e_2: $14 + 2 + 0 = 16 > K \Rightarrow$ cross it.
e_1: $2 + 2 + 2 = 6 \leq K \Rightarrow$ **explanation**.
Only e_1 is needed in the explanation. The
basic explanation would have added both.

Fig. 2. Example of improved explanations. The labels for propagation ('$\triangleright$', from Dijkstra's algorithm) and explanation ('$\triangleleft$', from Algorithm 2) are given next to each node.

Algorithm 2 mimics Dijkstra's starting at n in $\mathcal{G}^{-1}$. For any arc $e = (v, u)$ of weight $w[e]$ we know δ_v (obtained during propagation). We also have the distance from u to n (the cost of the last current node in the loop, line 4). Let X be the initially empty set of arcs explaining why n is at distance more than K from s. When considering a forbidden arc, if $\delta_v + w[e] + \delta_u^{-1} \leq K$, e participates in a path from s to n no longer than K. Therefore we add it to the explanation **and we do not cross it**. Otherwise, we can cross it. Once we dequeue s we finish since all other paths are no shorter than $\delta_s^{-1} > K$. See Fig. 2 for an example of explanation.

Theorem 2. *The clause $[\![\delta_d \leq K]\!] \wedge \bigwedge_{e_f \in X} \neg c_{e_f} \Rightarrow \neg c_n$ computed by Algorithm 2 is a correct and **minimal** explanation for why n is too far from s to be in G.*

Proof. Let F be the set of forbidden arcs at the time of explanation. At any stage of Algorithm 2, let F_p be the set of forbidden arcs not yet considered (initially F), X the arcs in the explanation, $d_{G'}(u, v)$ the shortest distance from u to v for any $G' \subseteq \mathcal{G}$, and u the top of Q. Let $G_R = \mathcal{G} \setminus (F_p \cup X)$. We ensure correctness and minimality by preserving the following invariants: (1) $d_{G_R}(s, n) > K$, (2) for all $(v', u') \in F_p$, $d_{G_R}(u, n) \leq d_{G_R}(u', n)$, and (3) for all $e \in X$, $d_{G_R \cup \{e\}}(s, n) \leq K$.

The three invariants hold initially: $G_R = \mathcal{G} \setminus F = G$, so (1) is the bound to be explained, (2) holds because n is initially the head of Q and all weights are non-negative, and (3) holds because X is initially empty.

At each iteration, we remove u from Q and process each arc $e = (v, u) \in \mathcal{E}$ (removing all forbidden arcs (v, u) from F_p, preserving (2) as nodes are processed in order of distance from n). We add arcs such that $\delta_v + w[e] + \delta_u^{-1} \leq K$ to X (preserving property (3)). Other forbidden arcs are now made available in G_R.

We show how adding these arcs to G_R maintains the invariants. Note $d_{G_R}(x, n)$ values for previously processed nodes x remain unchanged as any newly introduced path from n must be at least as long as δ_u^{-1}. Newly available arcs may, however, decrease $d_{G_R}(x, n)$ for some x which has not yet been processed. However, if $d_{G_R}(x, n)$ decreased as a result of (v, u) becoming available, then the shortest path from x to n must pass through u. But, x is not yet processed, so still $d_{G_R}(u, n) \leq d_{G_R}(x, n)$, preserving property (2). If the shortest path from s to n were to be reduced because now $\delta_x + d_{G_R}(x, n) < \delta_n$, there is

a contradiction since this path goes through (v, u) meaning the arc should have been added to X and be unavailable. Hence property (1) is preserved. Adding arcs to G_R can only make paths shorter, hence property (3) is preserved.

Once s is popped from Q, our explanation is X. By (1), $F_p \cup X$ is a valid explanation; but by (2), no arc e remaining in F_p may be on a shorter path from s to n (as either n is unreachable via e, or the head of the arc is distance no less than δ_s^{-1} from n), so e may be omitted from the explanation. Thus X is also a valid explanation. Removing arcs from F_p, thus adding them to G_R preserves property (3). By (3), omitting any element of X introduces a path from s to n of length no greater than K, so X is also minimal. $\qquad\square$

Clearly, Algorithm 2 runs in $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}|log(|\mathcal{V}|))$, like Dijkstra's algorithm. We can use it to explain Propositions 1 and 2 as follows. For Proposition 1, we first obtain $X_1 = \text{ExplainDist}(\mathcal{G}, s, u, \{\delta_v | v \in \mathcal{V}\}, K - \delta_u^{-1})$, the explanation for u being at distance $K - \delta_u^{-1}$ from s. The call to Algorithm 2 also yields the distance δ_u^* from s to u in $\mathcal{G}$ that is still greater than $K - \delta_u^{-1}$. Let X_2 be the explanation for d being at distance $K - \delta_u^*$ from u. The final explanation is $X = X_1 \cup X_2$. The same idea can be used for Proposition 2, using the head and tail of the arc to be removed.

DP-based Propagation. We can also improve the explanations for the DP-based propagation. Similarly to the simple propagation, Algorithm 1 leaves a table on each node stating the cost of visiting some subsets of mandatory nodes before getting to that node. If d is not reachable in less than $K + 1$ visiting all mandatory nodes, we fail and explain the failure. To do so, we run the same Algorithm 1 starting at d on the revered graph allowing forbidden arcs (similarly to Algorithm 2).

Let $e^{-1} = (v, u)$ be some reversed forbidden arc of cost $w[e]$. On node u (the tail of e in the original graph) lies the table left from the propagation pass of Algorithm 1. Each row of the table is a pair (m_u, γ_u) as defined in Sect. 4.3. Symmetrically, node v contains a table where each row (m_v, γ_v) indicates the mandatory nodes visited from d to v. If there exists an entry (m_u, γ_u) in u's table and an entry (m_v, γ_v) in v's table such that $m_v \cup m_u = M$, then e is an arc that could be used in a path from s to d containing all nodes in M. If additionally, $\gamma_v + w[e] + \gamma_u \leq K$, that path would be a valid path. Therefore, e being forbidden explains why we can't reach d visiting all mandatory nodes in no more than K. This corresponds to substituting the if-condition in line 8 of Algorithm 1 with a call to Explain from Algorithm 3 (where e is the reversed arc of whom we are visiting the tail, namely v).

State explosion for explanations: The explanation algorithm needs to use the same mandatory nodes as the propagation. Therefore, if we clustered, the same clustering is given to this algorithm. Also, we cannot use SCC levels here (other than the ones computed at the root) since we need to traverse forbidden arcs whether or not they skip entire mandatory SCCs as there may be other forbidden arcs leading to the skipped SCCs later.

A major problem with these explanations is that we need to traverse forbidden arcs. In dense graphs, this can be slow as there may be many possible paths to consider. For this reason, we use a simple stopping condition. Let t_p be the time it takes to run Algorithm 1 for propagation. If explaining is taking more than $x \times t_p$ (we choose x arbitrarily) we switch to the version of EXPLAIN in Algorithm 4 which corresponds to the basic explanations described in Theorem 1. We say that we *interrupt the explanation* when this change happens.

Algorithm 3. Better explanations	**Algorithm 4.** Avoiding state explosion
1: **function** EXPLAIN(e, γ_v, m_v)	1: **function** EXPLAIN(e, γ, m)
2: **for all** $(m_h, c_h) \in table[head(e)]$ **do**	2: ▷ $was\_short(e) = true \Leftrightarrow e$ was in a short-enough path at some point.
3: **if** $m_h \cup m_v = M$ **then**	
4: **if** $\gamma_h + w[e] + \gamma_v \leq K$ **then**	3: **if** $\neg c_e \wedge was\_short(e)$ **then**
5: $explanation.add(\neg c_e)$	4: $explanation.add(\neg c_e)$
6: **return** true	5: **return** true
7: **return** false	6: **return** false

5   Experimental Results

In this section we test our *bounded_path* in different problems (all benchmarks available at [4]). We implemented all our work in the CHUFFED solver [3]. All tests are run on a Linux 3.16 Intel® Core™ i7-4770 CPU @ 3.40GHz machine.

We annotate the tests EXPL when learning is enabled, NOEXPL otherwise. We use EXPL* for the improved explanations. We name the tree decomposition for *path* PATH, BPATH the *bounded_path* propagator without the DP algorithm, and DPBPATH when using the DP algorithm. We compare failures (the number of times the solver has encountered a wrong valuation of the variables before proving optimality), the number of nodes (the size of the search space explored) and the time in seconds.

We found it beneficial to add an array of successors constrained as $c_e \Leftrightarrow succ[tail(e)] = head(e)$. Definitions of all search strategies are given in [15].

5.1   Node Constrained Shortest Paths

Here we compare our path propagators with the results from [17] using their same benchmarks. The aim of these problems is to find the shortest path between two given nodes in a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ passing through a set of mandatory nodes M. We present the results in Table 1 using `first_fail` on the *succ* variables as the search strategy.

We clearly see that we solved the benchmarks faster than in [17]. We also see how BPATH and DPBPATH improve the results obtained by PATH, which is the point of having *bounded_path*. We can also see that the explanations reduced the number of failures greatly, specially for the two instances with biggest search

Table 1. Comparison between [10,17], Expl, NoExpl, BPath, DPBPath and Path. (C) indicates when clustering is used.

Benchmark	$|\mathcal{N}|$	[17] Fails	Time(s)	[10] Fails	Time(s)		PATH Fails	Time(s)	PATH+BPATH Fails	Time(s)	PATH+DPBPATH Fails	Time(s)
Ham22	22	13	4.45	24	0.00	Expl	139	0.03	19	0.01	16	0.01
Ham22full	22	0	1.22	2	0.00		19	0.01	15	0.01	15	0.01
Ham52b	52	100	402	112	0.01		1119	0.81	19	0.07	19	0.22
Ham52full	52	3	45.03	5	0.00		90	0.13	72	0.11	72	0.58 (C)
Ham52order_a	52	16	57.07	97	0.02		2203	2.54	189	0.45	76	3.80
Ham52order_b	52	41	117	1	0.00		49	0.04	49	0.05	49	0.08
See [17] for details on						NoExpl	202	0.02	34	0.01	22	0.01
on the benchmarks.							35	0.01	27	0.01	13	0.74
"full" $\equiv M = \mathcal{N}$							17579	6.04	1523	0.76	21	4.03
"order" $\equiv$ the nodes							328	0.12	264	0.12	264	0.59(C)
in M must be visited							17438	7.93	1409	0.83	407	0.38
in some given order.							83	0.03	83	0.03	83	0.13

space (52b and 52order_a). We do not show Expl* as they don't improve on Expl, because the search space is already very small, and Expl* is more expensive than Expl.

Although slow, the DPBPath is still suitable for the Hamiltonian path of 22 nodes. For 52 nodes in such dense graph though, the state space explodes and we absolutely need to cluster.

We also compared against the circuit-based path propagator with explanations presented in [10]. Their propagator is surprisingly fast on these benchmarks and requires little search. This is because their propagator has much better reasoning over the topology of the graph. The topological reasoning of our case is done by the *path* propagator, which is a combination of directed trees (Eq. 1), whereas their propagator makes more inferences based on strongly connected components and starting the path at different nodes. This specific benchmarks are simple in terms of distance (all the arcs have the same weight), but hard in terms of topology, hence the advantage.

The take-away from this experiment is that for graphs that are topologically hard, using our propagator might be a burden whereas using other propagators with strong topological reasoning as [10] might be a better approach.

5.2   Metabolic Networks

A metabolic network is a network of molecules and reactions. Biologists use this to understand how some molecules transform into others and cause some behavior in cells. For instance, this helps biologists understand how a protein behaves or how gene expression is regulated. This problem was modeled in [25] by creating a bipartite graph where molecules are in one partition of the nodes, and reactions in another partition. The arcs of the graph link the substrates and products participating in a reaction to the reaction itself.

Here, there is a set of mandatory nodes (because biologists are aware of their existence) and mutually exclusive nodes (corresponding to mutually exclusive reactions). Furthermore, each node is given a weight corresponding to its degree

(this is to model highly connected molecules). The objective is to find a pathway from some given substrate to some given product minimizing the total weight of the path, where the weight is the sum of the degrees of the nodes.

Table 2 shows a comparison between our solver and the work in [25], which used the solvers GRASPER and CP(Graph) on an Intel Core 2 Duo 2.16 GHz. Here BPATH stands for PATH+BPATH. There is one instance for each size.

Table 2. Solving metabolic pathways in real-world networks (same strategy as [25]).

	Glycosis				Lysine				Heme			
$|\mathcal{N}|$	GRASP.	CP (Graph)	PATH	BPATH	GRASP.	CP (Graph)	PATH	BPATH	GRASP.	CP (Graph)	PATH	BPATH
500	0.28	0.21	**0.05**	0.11	0.36	0.41	**0.06**	0.12	0.22	0.10	**0.05**	0.22
600	0.38	0.31	**0.07**	0.17	0.48	0.44	**0.06**	0.16	0.28	0.12	**0.06**	0.31
700	0.45	0.35	**0.19**	0.22	0.47	0.75	**0.08**	0.25	0.36	0.16	**0.08**	0.46
800	0.53	0.50	**0.24**	0.29	0.53	1.00	**0.12**	0.37	0.41	0.19	**0.11**	0.55
900	0.64	0.68	**0.15**	0.39	0.57	1.29	**0.16**	0.4	0.51	0.27	**0.15**	0.73
1000	0.77	0.84	**0.18**	0.51	0.60	1.37	**0.18**	0.46	0.62	0.32	**0.18**	0.95
1100	0.91	1.00	**0.17**	0.71	0.73	1.29	**0.19**	0.64	0.65	0.33	**0.32**	1.08
1200	0.96	1.08	**0.20**	0.75	0.86	2.23	**0.23**	0.79	0.80	0.41	**0.21**	3.62
1300	1.03	1.21	**0.81**	0.84	0.99	2.50	**0.28**	1.02	0.94	0.47	**0.4**	1.81
1400	1.23	1.56	**0.71**	1.05	1.12	2.84	**0.30**	1.17	1.11	0.51	**0.4**	2.1
1500	1.40	1.85	**1.25**	1.28	1.25	2.92	**0.39**	1.33	1.14	**0.52**	0.94	2.09
1600	1.67	2.14	**0.75**	1.49	1.30	2.97	**0.43**	1.36	1.35	**0.61**	0.74	2.55
1700	1.93	2.40	**0.82**	1.77	1.41	3.03	**0.67**	1.44	1.57	0.69	**0.4**	3.08
1800	2.11	2.77	**1.01**	2.01	1.53	3.69	**0.49**	1.69	1.72	0.77	**0.45**	3.69
1900	2.27	3.02	**1.19**	2.21	1.75	3.93	**0.60**	1.95	1.96	0.84	**0.48**	6.21
2000	2.40	3.14	**1.33**	2.3	1.96	2.18	**0.64**	2.39	2.18	0.91	**0.51**	4.86

The results show that BPATH slows PATH down. We interpret this as the effect of the overhead of *bounded_path*. Indeed, the instances are solved so quickly by PATH that BPATH has little to improve on. We also ran the same experiments with the VSIDS [14] search strategy. The times were very similar to those in Table 2 for PATH, but 3 benchmarks (1200, 1300 and 1900 nodes for Heme) were much slower (around 30 seconds). We tested the BPATH version on those three instances and noticed a big speedup (between 5 and 15 times faster). Nonetheless, note how BPATH is still faster than GRASPER and CP(Graph) in two thirds of the tests. From this we conclude that bounding is only worthwhile if the instances are hard to solve (i.e. there is a big search space to explore).

5.3 Task Constrained Shortest Path

In this problem, we are required to perform a set of tasks along a path. A task can be done at different nodes, and visiting a node where some task can be performed is enough, we do not need to visit more than one. As an example, consider on the drive home withdrawing money from an ATM, going to a carwash

and buying some groceries. Any ATM, supermarket or carwash on the path is sufficient. This problem was studied in [18, 19] using dynamic programming only.

In [10], the authors used a circuit-based path propagator to solve a similar problem (minimizing the longest arc). We compare our implementation against theirs using the same instances (500 graphs of 20 nodes each) with the objective of minimizing the total length of the path. The aim of this experiment is to see if BPath and DPBPath can also improve the circuit-based path propagator.

In this experiment we compare the best runtimes of both approaches, even if they use two different strategies. Our best search strategy is `smallest` (i.e. branching on the *succ* variable with smallest domain) and their best search strategy is `first_fail` on the *succ* variables. Additionally, we combine our bounding propagator with theirs to see the benefits.

Table 3. Benefit from BPath &DPBPath with both Path and [10]. Geometric average over 500 instances of 20 nodes.

Version	EXPL* (all use `smallest`)				EXPL				NoEXPL			
	Conflicts	Nodes	Time	Opt	Conflicts	Nodes	Time	Opt	Conflicts	Nodes	Time	Opt
		Nodes	(s)	(s)			(s)	(s)			(s)	(s)
[10]	48790	54254	3.18	2.14	48790	54254	3.18	2.14	308888	619304	7.95	6.48
[10]+BPath	18303	19883	2.90	1.49	27050	29995	3.70	2.84	174329	350327	15.99	13.67
[10]+DPBPath	636	1133	2.09	1.86	4933	6228	1.68	1.36	31256	188278	4.47	3.75
Path	26488	28801	7.05	2.27	26488	28801	7.05	2.27	200773	402943	32.63	9.81
Path+BPath	13175	14787	3.63	1.30	15238	16868	4.07	1.37	76701	156208	16.20	5.51
Path+DPBPath	**54**	**456**	**0.53**	**0.36**	**221**	**648**	**1.31**	**0.44**	**381**	**1253**	**2.96**	**1.14**

Table 3 gives the results, also showing the time (Opt) to find (but not prove) the optimal solution. The Path constraint finds optimal solutions very fast, but takes time to prove optimality. On the other hand, the version from [10] is superior in both these aspects. Adding BPath and DPBPath improves both these versions. The circuit based propagator does 89 % less search when combined with DPBPath (in its fastest version, using EXPL), and Path does 98 % less search when combined with DPBPath (using EXPL*). This shows how *bounded_path* with explanations can be used in combination with both tree-based and circuit-based paths to enhance propagation.

5.4   Profitable Tourist Path

We introduce here a new problem (as far as we are aware) similar to the prize collecting TSP. Imagine you need to do a long layover during a trip and change airports. You might be interested in visiting the city while waiting for your connection flight. In this problem, we model every point of interest (POI) of a city with a minimum visit time (i.e. the least amount of time that a visit to some POI is worthwhile) and a profit (i.e. how much a person enjoys visiting some POI). The path can contain a node without necessarily visiting the corresponding POI, but in order to visit a POI the path must contain the corresponding node

and spend the minimum visit time. The objective of the problem is to find the path with most profit such that the total time is less than a certain bound (i.e. the time we have available between connections). The total time is the cost of the path plus the time spent at each POI (either 0 or the minimum visit time).

We created two benchmarks, based on New York City (14 nodes, from LGA Airport to JFK Airport) and London (12 nodes, from Heathrow Airport to Liverpool Street Station). We added two side constraints: for London, we require that the visit to the Tower Bridge (if it happens) takes place between two narrow time frames (which would correspond to times where the bridge opens to let ships go through); for NYC, the ferry to Liberty Island leaves every hour and so there might be a waiting time added to the total time (if the visit happens).

We used EXPL on all the tests to study the benefits of bounding. The results are in Table 4. Clearly, DPBPATH and BPATH largely improve PATH for this problem. Again, there was no need to cluster or interrupt explanations.

Table 4. Profitable tourist path. Search: `smallest` on *succ* variables.

Version	New York City (14 POI)			London (12 POI)		
	Fails	Nodes	Time(s)	Fails	Nodes	Time(s)
PATH	≥5030898	>5034046	>3600.00	236010	237263	60.14
PATH+BPATH (EXPL)	390985	391746	379.19	24061	25190	13.86
PATH+DPBPATH (EXPL)	44015	44606	48.82	10645	11866	2.89
PATH+BPATH (EXPL*)	360945	361971	350.26	18546	19881	8.78
PATH+DPBPATH (EXPL*)	**2062**	**2690**	**37.45**	**224**	**670**	**0.16**

Without explanations, though, NYC takes 1598 s using DPBPATH and London takes 13 s, making them substantially slower than with explanations.

6   Conclusion

In this paper we have improved the *bounded_path* propagator by adding a new propagation technique that is clearly superior. Both propagations are enhanced by our two new versions of explanations. First, a fast way of computing valid but not minimal explanations is given. We then provided another version that generates more reusable explanations.

We have shown how combining *bounded_path* with path propagators (composition of directed trees or circuit-based) improves their performance, reaching the state of the art in bounded path propagation.

References

1. Beldiceanu, N., Flener, P., Lorca, X.: The *tree* constraint. In: Barták, R., Milano, M. (eds.) CPAIOR 2005. LNCS, vol. 3524, pp. 64–78. Springer, Heidelberg (2005)
2. Beldiceanu, N., Katriel, I., Lorca, X.: Undirected forest constraints. In: Beck, J.C., Smith, B.M. (eds.) CPAIOR 2006. LNCS, vol. 3990, pp. 29–43. Springer, Heidelberg (2006)
3. Chu, G.G.: Improving combinatorial optimization. Ph.D. thesis. The University of Melbourne (2011)
4. De Uña, D.: Directed graph benchmarks (2015). http://people.eng.unimelb.edu.au/pstuckey/bounded_path/bounded_path.zip
5. De Uña, D., Gange, G., Schachte, P., Stuckey, P.J.: Steiner tree problems with side constraints using constraint programming. In: Proceedings of the Thertieth AAAI Conference on Artificial Intelligence. AAAI Press (2016, to appear)
6. Dooms, G., Deville, Y., Dupont, P.E.: CP(Graph): introducing a graph computation domain in constraint programming. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 211–225. Springer, Heidelberg (2005)
7. Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica **11**(4), 379–403 (1994)
8. Fages, J.G.: Exploitation de structures de graphe en programmation par contraintes. Ph.D. thesis. École de Mines de Nantes (2014)
9. Fages, J.-G., Lorca, X.: Revisiting the *tree* constraint. In: Lee, J. (ed.) Principles and Practice of Constraint Programming – CP 2011. LNCS, vol. 6876, pp. 271–285. Springer, Heidelberg (2011)
10. Francis, K.G., Stuckey, P.J.: Explaining circuit propagation. Constraints **19**(1), 1–29 (2014)
11. Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm. J. R. Stat. Soc. Ser. C (Applied Statistics) **28**(1), 100–108 (1979)
12. Hopcroft, J.E., Ullman, J.D., Aho, A.V.: Data structures and algorithms, vol. 175. Addison-Wesley Boston, USA (1983)
13. Lengauer, T., Tarjan, R.E.: A fast algorithm for finding dominators in a flowgraph. ACM Trans. Program. Lang. Syst. (TOPLAS) **1**(1), 121–141 (1979)
14. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient sat solver. In: Proceedings of the 38th annual Design Automation Conference, pp. 530–535. ACM (2001)
15. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.R.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)
16. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009). http://dx.doi.org/10.1007/s10601-008-9064-x
17. Quesada, L., Van Roy, P., Deville, Y., Collet, R.: Using dominators for solving constrained path problems. In: Hentenryck, P. (ed.) PADL 2006. LNCS, vol. 3819, pp. 73–87. Springer, Heidelberg (2005)
18. Rice, M.N., Tsotras, V.J.: Engineering generalized shortest path queries. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 949–960. IEEE (2013)
19. Rice, M.N., Tsotras, V.J.: Parameterized algorithms for generalized traveling salesman problems in road networks. In: Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 114–123. ACM (2013)

20. Sellmann, M.: Cost-based filtering for shorter path constraints. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 694–708. Springer, Heidelberg (2003)
21. Sellmann, M., Gellermann, T., Wright, R.: Cost-based filtering for shorter path constraints. Constraints **12**(2), 207–238 (2006). http://dx.doi.org/10.1007/s10601-006-9006-4
22. Silver, D.: Cooperative pathfinding. In: AIIDE, pp. 117–122 (2005)
23. Storandt, S.: Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In: Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science, pp. 20–25. ACM (2012)
24. Storandt, S.: Route planning for bicycles-exact constrained shortest paths made practical via contraction hierarchy. In: ICAPS, vol. 4, p. 46 (2012)
25. Viegas, R.D., Azevedo, F.: Lazy constraint imposing for improving the path constraint. Electron. Notes Theor. Comput. Sci. **253**(4), 113–128 (2009)

Compact-Table: Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets

Jordan Demeulenaere[1], Renaud Hartert[1], Christophe Lecoutre[2],
Guillaume Perez[3], Laurent Perron[4], Jean-Charles Régin[3],
and Pierre Schaus[1(✉)]

[1] UCLouvain, Louvain-la-Neuve, Belgium
pschaus@gmail.com
[2] CRIL, Univ. Artois and CNRS, 62300 Lens, France
[3] University of Nice, Nice, France
[4] Google, Paris, France

Abstract. In this paper, we describe Compact-Table (CT), a bitwise algorithm to enforce Generalized Arc Consistency (GAC) on table constraints. Although this algorithm is the default propagator for table constraints in or-tools and OscaR, two publicly available CP solvers, it has never been described so far. Importantly, CT has been recently improved further with the introduction of residues, resetting operations and a data-structure called reversible sparse bit-set, used to maintain tables of supports (following the idea of tabular reduction): tuples are invalidated incrementally on value removals by means of bit-set operations. The experimentation that we have conducted with OscaR shows that CT outperforms state-of-the-art algorithms STR2, STR3, GAC4R, MDD4R and AC5-TC on standard benchmarks.

1 Introduction

Table constraints, also called extension(al) constraints, explicitly express the allowed combinations of values for the variables they involve as sequences of tuples, which are called tables. Table constraints can theoretically encode any kind of constraints and are among the most useful ones in Constraint Programming (CP). Indeed, they are often required when modeling combinatorial problems in many application fields. The design of filtering algorithms for such constraints has generated a lot of research effort, see [1,10,12,17,20,21,23,30,33].

Over the last decade, many developments have thus been achieved for enforcing the well-known property called Generalized Arc Consistency (GAC) on binary and/or non-binary extensionally defined constraints. Among successful techniques, we find:

– bitwise operations that allow performing parallel operations on bit vectors. Already exploited during the 70's [27,32], they have been applied more recently to the enforcement of arc consistency on binary constraints [3,22].

– residual supports (residues) that store the last found supports of each value. Initially introduced for ensuring optimal complexity [2], they have been shown efficient in practice [18,19,24] when used as simple sentinels.
– tabular reduction, which is a technique that dynamically maintains the tables of supports. Based on the structure of sparse sets [4,16], variants of Simple Tabular Reduction (STR) have been proved to be quite competitive [17,20,33].
– resetting operations that saves substantial computing efforts in some particular situations. They have been successfully applied to the algorithm GAC4 [30].

In this paper, we introduce a very efficient GAC algorithm[1] for table constraints that combines the use of bitwise operations, residual supports, tabular reduction, and resetting operations. It is called Compact-Table (CT), and originates from or-tools, the Google solver that won the latest MiniZinc Challenges. It is important to note that or-tools developers prefer to focus on highly-optimized implementations of a few important (global) constraints instead of having many of them. Through the years, CT has reached a good level of maturity because it has been continuously improved and extended with many cutting edge ideas such as those introduced earlier. Unfortunately, the core algorithm of CT has not been described in the literature so far[2] and is thus seldom used as a reference for practical comparisons. The first version of CT implemented in or-tools, with a bit-set representation of tables, dates back to 2012, whereas the version of CT presented in this paper is exactly the last one implemented in OscaR [29].

Outline. After presenting related works in Sect. 2, we introduce some technical background in Sect. 3. Then, we recall in Sect. 4 usual state restoration mechanisms implemented in CP solvers, and describe reversible sparse bit-sets in Sect. 5. In Sect. 6, we describe our algorithm CT. Before concluding, we present in Sect. 7 the results of an experimentation we have conducted with CT and its contenders on a large variety of benchmarks.

2   Related Work

Propagators for table constraints are filtering procedures used to enforce GAC. Given the importance of table constraints, it is not surprising that much research has been carried out in order to find efficient propagators. This section briefly describes the most efficient ones.

Generic Algorithms. On the one hand, GAC3 is a classical general-purpose GAC algorithm [25] for non-binary constraints. Each call to this algorithm for a constraint requires testing if each value is still supported by a valid tuple accepted by the constraint. Several improvements to fasten the search for a support gave birth to variants such as GAC2001 [2] and GAC3^{rm} [19]. Unfortunately, the

---

[1] We are aware of an independent work [34] on a similar topic, but hadn't the opportunity of reading it at the time of writing our paper.
[2] Note that some parts of this paper were published in a Master Thesis report [7].

worst-case time complexity of all these algorithms grows exponentially with the arity of the constraints. On the other hand, GAC4 [28] is a value-based algorithm, meaning here that for each value, it maintains a set of valid tuples supporting it. Each time a value is removed, all supporting tuples are removed from the associated sets, which allows us to identify values without any more supports. GAC4R is a recent improvement of GAC4 [30], which recomputes the sets of supporting tuples from scratch (referred to as a resetting operations) when it appears to be less costly than updating them based on the removed values.

AC5 Instantiations. In [12], Mairy et al. introduce several instantiations of the generic AC5 algorithm for table constraints, the best of them being AC5-TCOptSparse. This algorithm shares some similarities with GAC4 since it precomputes lists of supporting tuples which allows us to retrieve efficiently new supports by iterating over these lists. Note that a reversible integer, i.e., an integer storage location with a facility to restore its successive values, is used to indicate the current position of a support in each list. This algorithm is implemented in Comet, and has been shown to be efficient on ternary and quaternary constraints.

Simple Tabular Reduction. STR1 [33] and STR2 [17] are algorithms that globally enforce GAC by traversing the constraint tables while dynamically maintaining them: each call to the algorithm for a constraint removes the invalid tuples from its table. The improvements brought in STR2 avoid unnecessary operations by considering only relevant subsets of variables when checking the validity of a tuple, and collecting supported values. Contrary to its predecessors, STR3 [20] is a fine-grained (or value-based) algorithm. For each value, it initially computes a static array of tuples supporting it, and keeps a reversible integer *curr* that indicates the position of the last valid tuple in the array. STR3 also maintains the set of valid tuples. STR3 is shown to be complementary to STR2, being more efficient when the tables are not reduced drastically during search.

Compressed Representations. Other algorithms gamble on the compression of tables to reduce the time needed to ensure GAC. The most promising data structure allowing a more compact representation is the Multi-valued Decision Diagram (MDD) [31], but note that the order of variables used to build an MDD may significantly impact its size. Two notable algorithms using MDDs as main data structure are `mddc` [6] and MDD4R [30]. The former does not modify the decision diagram and performs a depth-first search of the MDD during propagation to detect which parts of the MDD are consistent or not. MDD4R dynamically maintains the MDD by deleting nodes and edges that do not belong to a solution. Each value is matched with its corresponding edges in the MDD, so, when a value has none of its edges present in the MDD, it can be removed. On the other hand, some other forms of compression have been studied from the concepts of compressed tuples [14,35], short supports [13] and sliced tables [11].

3   Technical Background

A *constraint network* (CN) N is composed of a set of n variables and a set of e constraints. Each *variable* x has an associated domain, denoted by $dom(x)$, that contains the finite set of values that can be assigned to it. Each *constraint* c involves an ordered set of variables, called the *scope* of c and denoted by $scp(c)$, and is semantically defined by a *relation*, denoted by $rel(c)$, which contains the set of tuples allowed for the variables involved in c. The arity of a constraint c is $|scp(c)|$, i.e., the number of variables involved in c. A (positive) *table constraint* c is a constraint such that $rel(c)$ is defined explicitly by listing the tuples that are allowed by c.

Example 1. The constraint $x \neq y$ with $x \in \{1, 2, 3\}$ and $y \in \{1, 2\}$ can be alternatively defined by the table constraint c such that $scp(c) = \{x, y\}$ and $rel(c) = \{(1, 2), (2, 1), (3, 1), (3, 2)\}$. We also write:

$$\langle x, y \rangle \in T \quad \text{with} \quad T = \langle (1, 2), (2, 1), (3, 1), (3, 2) \rangle$$

Let $\tau = (a_1, a_2, \ldots, a_r)$ be a tuple of values associated with an ordered set of variables $X = \{x_1, x_2, \ldots, x_r\}$. The ith value of τ is denoted by $\tau[i]$ or $\tau[x_i]$. The tuple τ is valid iff $\forall i \in 1..r, \tau[i] \in dom(x_i)$. An r-tuple τ is a *support* on the r-ary constraint c iff τ is a valid tuple that is allowed by c. If τ is a support on a constraint c involving a variable x and such that $\tau[x] = a$, we say that τ is a *support for* (x, a) on c. Generalized Arc Consistency (GAC) is a well-known domain-filtering consistency defined as follows:

Definition 1. *A constraint c is* generalized arc consistent *(GAC) iff $\forall x \in scp(c), \forall a \in dom(x)$, there exists at least one support for (x, a) on c. A CN N is GAC iff every constraint of N is GAC.*

Enforcing GAC is the task of removing from domains all values that have no support on a constraint. Many algorithms have been devised for establishing GAC according to the nature of the constraints. For table constraints, STR [33] is such an algorithm: it removes invalid tuples during search of supports using a sparse set data structure which separates valid tuples from invalid ones. This method of seeking supports improves search time by avoiding redundant tests on invalid tuples that have already been detected as invalid during previous GAC enforcements. STR2 [17], an optimization of STR, limits some operations concerning the validity of tuples and the identification of supports, through the introduction of two sets called S^{sup} and S^{val} (described later in Sect. 6).

4   Reversible Objects and Implementation Details

Trail and Timestamping. The issue of storing related states of the solving process is essential in CP. In many solvers[3], a general mechanism is used for doing and undoing (on backtrack) the current state. This mechanism is called a trail and

---

[3] One notable exception is Gecode, a copy-based solver.

it was first introduced in [9] for implementing non-deterministic search. A trail is a stack of pairs (*location*, *value*) where *location* stands for any piece of memory (e.g., a variable), which can be restored when backtracking. Typically, at each search node encountered during the solving process, the constraint propagation algorithm is executed. A same filtering procedure (propagator) can thus be executed several times at a given node. Consequently, if one is interested in storing some information concerning a filtering procedure, the value of a same memory location can be changed several times. However, stamping that is part of the "folklore" of programming [15] can be used to avoid storing a same memory location on the trail more than once per search node. The idea behind timestamping is that only the final state of a memory location is relevant for its restoration on backtrack. The trail contains a general time counter that is incremented at each search node, and a timestamp is attached to each memory location indicating the time at which its last storage on the trail happened. If a memory location changes and its timestamp matches the current time of the trail then there is no need to store it again. CP solvers generally expose some "reversible" objects to the users using this trail+timestamping mechanism. The most basic one is the reversible version of primitive types such as int or long values. In the following, we denote by `rint` and `rlong` the reversible versions of `int` and `long` primitive types.

Reversible Sparse Sets. Reversible primitive types can be used to implement more complex data structures such as reversible sets. It was shown in [16] how to implement a reversible set using a single `rint` that represents the current size (limit) of the set. In this structure, which is called reversible sparse set, an array of size n is used to store the permutation from 0 to $n - 1$. All values in this permutation array at indices smaller than or equal to a variable *limit* are considered as part of the set, while the others are considered as removed. When iterating on current values of the set (with decreasing indices from *limit* to 0), the value at the current index can be removed in $O(1)$ by just swapping it with the value stored at *limit* and decrementing *limit*. Making a sparse set reversible just requires managing a single `rint` for *limit*. On backtrack, when the limit is restored, all concerned removed values are restored in $O(1)$.

Domains and Deltas. In OscaR [29], the implementation of domains relies on reversible sparse sets. One advantage is that one can easily retrieve the set of values removed from a domain between any two calls to a given filtering procedure. All we need to store in the filtering procedure is the last size of the domain. The delta set (set of values removed between the two calls) is composed of all the values located between the current size and the last recorded size. More details on this cheap mechanism to retrieve the delta sets can be found in [16].

5   Reversible Sparse Bit-Sets

This section describes the class `RSparseBitSet` that is the main data structure for our algorithm to maintain the supports. In what follows, when we refer to an array t, $t[0]$ denotes the first element (indexing starts at 0) and t.length the number of its cells (size). Also, 0^k will stand for a sequence of k bits set to 0.

Algorithm 1. Class RSparseBitSet

---

1 `words`: array of rlong   `// words.length = p`
2 `index`: array of int   `// index.length = p`
3 `limit`: rint
4 `mask`: array of long   `// mask.length = p`

5 Method isEmpty(): **Boolean**
6 $\quad$ **return** `limit` $= -1$

7 Method clearMask()
8 $\quad$ **foreach** i **from** 0 **to** `limit` **do**
9 $\quad\quad$ `offset` $\leftarrow$ `index`$[i]$
10 $\quad\quad$ `mask`$[$`offset`$] \leftarrow 0^{64}$

11 Method reverseMask()
12 $\quad$ **foreach** i **from** 0 **to** `limit` **do**
13 $\quad\quad$ `offset` $\leftarrow$ `index`$[i]$
14 $\quad\quad$ `mask`$[$`offset`$] \leftarrow$ ˜`mask`$[$`offset`$]$ `// bitwise NOT`

15 Method addToMask(**m: array of long**)
16 $\quad$ **foreach** i **from** 0 **to** `limit` **do**
17 $\quad\quad$ `offset` $\leftarrow$ `index`$[i]$
18 $\quad\quad$ `mask`$[$`offset`$] \leftarrow$ `mask`$[$`offset`$] — m[$`offset`$]$ `// bitwise OR`

19 Method intersectWithMask()
20 $\quad$ **foreach** i **from** `limit` **downto** 0 **do**
21 $\quad\quad$ `offset` $\leftarrow$ `index`$[i]$
22 $\quad\quad$ $w \leftarrow$ `words`$[$`offset`$]$ &`mask`$[$`offset`$]$ `// bitwise AND`
23 $\quad\quad$ `words`$[$`offset`$] \leftarrow w$
24 $\quad\quad$ **if** $w = 0^{64}$ **then**
25 $\quad\quad\quad$ `index`$[i] \leftarrow$ `index`$[$`limit`$]$
26 $\quad\quad\quad$ `index`$[$`limit`$] \leftarrow$ `offset`
27 $\quad\quad\quad$ `limit` $\leftarrow$ `limit` $- 1$

28 Method intersectIndex(**m: array of long**): **int**
$\quad$ `/* Post: returns the index of a word where the bit-set`
$\quad$ `intersects with m, −1 otherwise` $\quad\quad\quad\quad\quad\quad\quad$ `*/`
29 $\quad$ **foreach** i **from** 0 **to** `limit` **do**
30 $\quad\quad$ `offset` $\leftarrow$ `index`$[i]$
31 $\quad\quad$ **if** `words`$[$`offset`$]$ & $m[$`offset`$] \neq 0^{64}$ **then**
32 $\quad\quad\quad$ **return** `offset`
33 $\quad$ **return** -1

The class `RSparseBitSet`, which encapsulates four fields and 6 methods, is given in Algorithm 1. One important field is `words`, an array of p 64-bit words (actually, reversible long integers), which defines the current value of the bit-set:

the ith bit of the jth word is 1 iff the $(j-1) \times 64 + i$th element of the (initial) set is present. Initially, all words in this array have all their bits at 1, except for the last word that may involve a suffix of bits at 0. For example, if we want to handle a set initially containing 82 elements, then we build an array with $p = \lceil 82/64 \rceil = 2$ words that initially looks like:

```
words: 11111111111111111111111111111111 11111111111111111111111111111111
       11111111111111111100000000000000 00000000000000000000000000000000
```

Because, in our context, only non-zero words (words having at least one bit set to 1) are relevant when processing operations on the bit-set, we rely on the sparse-set technique by managing in an array `index` the indices of all words: the indices of all non-zero words are in `index` at positions less than or equal to the value of a variable `limit`, and the indices of all zero-words are in `index` at positions strictly greater than `limit`. For our example, we initially have:

```
words: 11111111111111111111111111111111 11111111111111111111111111111111
       11111111111111111100000000000000 00000000000000000000000000000000
index: 0 1
limit : 1
```

If we suppose now that the 66 first elements of our set above are removed, we obtain:

```
words: 00000000000000000000000000000000 00000000000000000000000000000000
       00111111111111111100000000000000 00000000000000000000000000000000
index: 1 0
limit: 0
```

The class invariant describing the state of a reversible sparse bit-set is the following:

– `index` is a permutation of $[0, \ldots, p-1]$, and
– `words[index[i]]` $\neq 0^{64} \Leftrightarrow i \leq$ `limit`, $\forall i \in 0..p-1$

Note that the reversible nature of our object comes from (1) an array of reversible long (denoted `rlong`) (instead of simple longs) to store the bit words, and (2) the reversible prefix size of non-zero words by using a reversible int (`rint`).

A `RSparseBitSet` also contains a local temporary array, called `mask`. Is is used to collect elements with Method addToMask(), and can be cleared and reversed too. A `RSparseBitSet` can only be modified by means of the method intersectWithMask() which is an operation used to intersect with the elements collected in `mask`. An illustration of the usage of these methods is given in next example.

words	1	0	1	0	1	1	1	1
addToMask	1	1	1	0	1	0	0	0
addToMask	0	0	0	1	0	0	0	1
mask	1	1	1	1	1	0	0	1
intersectWithMask	1	0	1	0	1	0	0	1

Fig. 1. RSparseBitSet example

Example 2. Figure 1 illustrates the use of Methods addToMask() and intersect WithMask(). We assume that the current state of the bit-set is given by the value of `words`, and that clearMask() has been called such that `mask` is initially empty. Then two bit-sets are collected in `mask` by calling addToMask(). The value of `mask` is represented after these two operations. Finally intersectWithMask() is executed and the new value of the bit-set `words` is given at the last row of Fig. 1.

We now describe the implementation of the methods in `RSparseBitSet`. Method isEmpty() simply checks if the number of non-zero words is different from zero (if the limit is set to -1, it means that all words are non-zero). Method clearMask() sets to 0 all words of `mask` corresponding to non-zero words of `words`, whereas Method reverseMask() reverses all words of `mask`. Method addToMask() applies a word by word logical bit-wise *or* operation. Once again, notice that this operation is only applied to words of `mask` corresponding to non-zero words of `words`. Method intersectMask() considers each non-zero word of `words` in turn and replaces it by its intersection with the corresponding word of `mask`. In case the resulting new word is zero, it is swapped with the last non-zero word and the value of `limit` is decremented. Finally, Method intersectIndex() checks if a given bit-set (array of longs) intersects with the current bit-set: it returns the index of the first word where an intersection can be proved, -1 otherwise.

6   Compact-Table (CT) Algorithm

As STR2 and STR3, Compact-Table (CT) is a GAC algorithm that dynamically maintains the set of valid supports regarding the current domain of each variable. The main difference is that CT is based on an object `RSparseBitSet`. In this set, each tuple is indexed by the order it appears in the initial table. Invalid tuples are removed during the initialization as well as values that are not supported by any tuple. The class `ConstraintCT`, Algorithm 2, allows us to implement any positive table constraint c while running the CT algorithm to enforce GAC.

6.1   Fields

As fields of Class `ConstraintCT`, we first find `scp` for representing the scope of c and `currTable` for representing the current table of c by means of a reversible sparse bit-set. If $\langle \tau_0, \tau_1, \ldots, \tau_{t-1} \rangle$ is the initial table of c, then `currTable` is a

`RSparseBitSet` object (of initial size t) such that the value i is contained (is set to 1) in the bit-set if and only if the ith tuple is valid:

$$i \in \texttt{currTable} \Leftrightarrow \forall x \in scp(c), \tau_i[x] \in dom(x)$$

We also have three fields $\texttt{S}^{\texttt{val}}$, $\texttt{S}^{\texttt{sup}}$ and `lastSizes` in the spirit of STR2. The set $\texttt{S}^{\texttt{val}}$ contains variables whose domains have been reduced since the previous invocation of the filtering algorithm on c. To set up $\texttt{S}^{\texttt{val}}$, we need to record the domain size of each modified variable x right after the execution of CT on c: this value is recorded in $\texttt{lastSizes}[x]$. The set $\texttt{S}^{\texttt{sup}}$ contains unfixed variables (from the scope of the constraint c) whose domains contain each at least one value for which a support must be found. These two sets allow us to restrict loops on variables to relevant ones.

We also have a field `supports` containing static data. During the set up of the table constraint c, CT also computes a static array of words $\texttt{supports}[x,a]$, seen as a bit-set, for each variable-value pair (x,a) where $x \in scp(c) \land a \in dom(x)$: the bit at position i in the bit-set is 1 if and only if the tuple τ_i in the initial table of c is a support for (x,a).

T	x	y	z
0	a	a	a
1	a	a	b
2	a	b	c
3	b	a	a
	a	c	b
4	a	b	b
5	b	a	b
6	b	b	a
7	b	b	b

currTable	1	1	1	1	1	1	1	1
supports$[x,a]$	1	1	1	0	1	0	0	0
supports$[x,b]$	0	0	0	1	0	1	1	1
supports$[y,a]$	1	1	0	1	0	1	0	0
supports$[y,b]$	0	0	1	0	1	0	1	1
supports$[y,d]$	0	0	0	0	0	0	0	0
supports$[z,a]$	1	0	0	1	0	0	1	0
supports$[z,b]$	0	1	0	0	1	1	0	1
supports$[z,c]$	0	0	1	0	0	0	0	0

(a) The indexed tuples   **(b)** The corresponding bit-sets

Fig. 2. Illustration of the data structures after the initialization of $\langle x, y, z \rangle \in T$. The tuple (a, c, b) will not be indexed and d will be removed from $dom(y)$.

Example 3. Figure 2 shows an illustration of the content of those bit-sets after the initialization of the following table constraint $\langle x, y, z \rangle \in T$, with:

– $dom(x) = \{a, b\}$, $dom(y) = \{a, b, d\}$, $dom(z) = \{a, b, c\}$
– $T = \langle (a, a, a), (a, a, b), (a, b, c), (b, a, a), (a, c, b), (a, b, b), (b, a, b), (b, b, a), (b, b, b) \rangle$

The tuple (a, c, b) is initially invalid because $c \notin dom(y)$, and thus will not be indexed. Value d will be removed from $dom(y)$ given that it is not supported by any tuple.

Finally, we have an array `residues` such that for each variable-value pair (x, a), $\texttt{residues}[x, a]$ denotes the index of the word where a support was found for (x, a) the last time one was sought for.

Algorithm 2. Class ConstraintCT

---

1 scp: array of variables   // Scope
2 currTable: RSparseBitSet   // Current table
3 S^{val}, S^{sup} // Temporary sets of variables
4 lastSizes   // lastSizes[x] is the last size of the domain of x
5 supports   // supports[x,a] is the bit-set of supports for (x,a)
6 residues   // residues[x,a] is the last found support for (x,a)

7 **Method** updateTable()
8   **foreach** *variable* $x \in S^{val}$ **do**
9   currTable.clearMask()
10   **if** $|\Delta_x| < |dom(x)|$ **then**   // Incremental update
11   **foreach** *value* $a \in \Delta_x$ **do**
12   currTable.addToMask(supports[x,a])
13   currTable.reverseMask()
14   **else**   // Reset-based update
15   **foreach** *value* $a \in dom(x)$ **do**
16   currTable.addToMask(supports[x,a])
17   currTable.intersectWithMask()
18   **if** currTable.isEmpty() **then**
19   **break**

20 **Method** filterDomains()
21   **foreach** *variable* $x \in S^{sup}$ **do**
22   **foreach** *value* $a \in dom(x)$ **do**
23   index $\leftarrow$ residues[x,a]
24   **if** currTable.words[index] & supports(x,a)[index] $= 0^{64}$ **then**
25   index $\leftarrow$ currTable.intersectIndex(supports[x,a])
26   **if** index $\neq -1$ **then**
27   residues[x,a] $\leftarrow$ index
28   **else**
29 $dom(x) \leftarrow dom(x) \setminus \{a\}$
30   lastSize[x] $\leftarrow |dom(x)|$

31 **Method** enforceGAC()
32 $S^{val} \leftarrow \{x \in$ scp $: |dom(x)| \neq$ lastSize[x]$\}$
33   **foreach** *variable* $x \in S^{val}$ **do**
34   lastSize[x] $\leftarrow |dom(x)|$
35 $S^{sup} \leftarrow \{x \in$ scp $: |dom(x)| > 1\}$
36   updateTable()
37   **if** currTable.isEmpty() **then**
38   **return** *Backtrack*
39   filterDomains()

6.2   Methods

The main method in `ConstraintCT` is enforceGAC(). After the initialization of the sets S^{val} and S^{sup}, CT updates `currTable` to filter out (indices of) tuples that are no more supports, and then considers each variable-value pair to check whether these values still have a support.

Updating the Current Table. For each variable $x \in S^{val}$, i.e., each variable x whose domain has changed since the last time the filtering algorithm was called, updateTable() performs some operations. This method assumes an access to the set of values Δ_x removed from $dom(x)$ since the last call to enforceGAC(). There are two ways of updating `currTable`, either incrementally or from scratch after resetting. Note that the idea of using resets has been proposed in [30] and successfully applied to GAC4 and MDD4, with the practical interest of saving computational effort in some precise contexts. This is the strategy implemented in updateTable(), by considering a reset-based computation when the size of the domain is smaller than the number of deleted values.

   In case of an incremental update (line 10), the union of the tuples to be removed is collected by calling addToMask() for each bit-set (of supports) corresponding to removed values, whereas in case of a reset-based update (line 14), we perform the union of the tuples to be kept. To get a mask ready to apply, we just need to reverse it when it has been built from removed values. Finally, the (indexes of) tuples of `currTable` not contained in the mask, built from x, are directly removed by means of intersectWithMask(). When there are no more tuples in the current table, a failure is detected, and updateTable() is stopped by means of a loop break.

Filtering of Domains. Values are removed from the domain of some variables during the search of a solution, which can lead to inconsistent values in the domain of other variables. As `currTable` is a reversible and dynamically maintained structure, the value of some bits changes from 1 to 0 when tuples become invalid (or from 0 to 1 when the search backtracks). On the contrary, the `supports` bit-sets are only computed at the creation of the constraint and are not maintained during search. It follows from the definition of those bit-sets that (x, a) has a valid support if and only if

$$(\texttt{currTable} \cap \texttt{supports}[x, a]) \neq \emptyset \tag{1}$$

   Therefore, each time a tuple becomes invalid, the constraint must check this condition for every variable value pair (x, a) such that $a \in dom(x)$, and remove a from $dom(x)$ if the condition is not satisfied any more. This operation is efficiently implemented in filterDomains() with the help of residues and the method intersectIndex().

Example 4. The same set of tuples as in Example 3 is considered. Suppose now that a was removed from $dom(x)$ (by another constraint) after the initialization.

Given that the domain of x is reduced, when updateTable() is called by enforce-GAC(), all tuples supporting a (because $\Delta_x = \{a\}$) will be invalidated. Figure 3a illustrates the intermediary bit-sets used to compute the new value $\texttt{currTable}^{out}$ from $\texttt{currTable}^{in}$ and $\texttt{supports}[x,a]$. Then filterDomains() computes for each variable-value pair (x_i, a_i) (with $x_i \in \texttt{S}^{\texttt{sup}}$ and $a_i \in dom(x)$) the intersection of its associated set of supports with $\texttt{currTable}$ as shown in Fig. 3b. Given that the intersection for $\texttt{supports}[z,c]$ and $\texttt{currTable}$ is empty, c is removed from $dom(z)$.

$\texttt{currTable}^{in}$	1	1	1	1	1	1	1	1
$\texttt{supports}[x,a]$	1	1	1	0	1	0	0	0
$\texttt{currTable}^{out}$	0	0	0	1	0	1	1	1

(a) updateTable() invalidates tuples supporting (x,a)

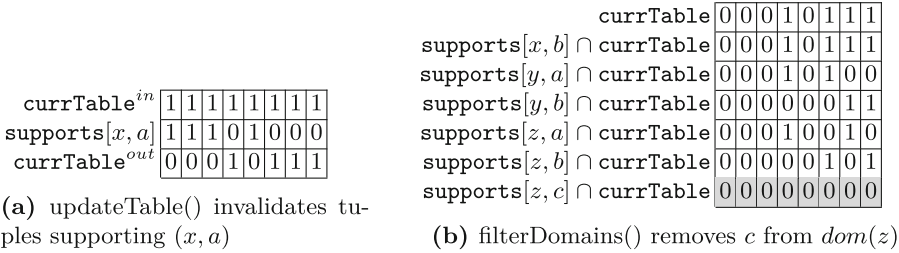|								
$\texttt{currTable}$	0	0	0	1	0	1	1	1
$\texttt{supports}[x,b] \cap \texttt{currTable}$	0	0	0	1	0	1	1	1
$\texttt{supports}[y,a] \cap \texttt{currTable}$	0	0	0	1	0	1	0	0
$\texttt{supports}[y,b] \cap \texttt{currTable}$	0	0	0	0	0	0	1	1
$\texttt{supports}[z,a] \cap \texttt{currTable}$	0	0	0	1	0	0	1	0
$\texttt{supports}[z,b] \cap \texttt{currTable}$	0	0	0	0	0	1	0	1
$\texttt{supports}[z,c] \cap \texttt{currTable}$	0	0	0	0	0	0	0	0

(b) filterDomains() removes c from $dom(z)$

Fig. 3. Illustration of enforceGAC() after the removal of a from $dom(x)$.

6.3   Improvements

The algorithm in Sect. 6.2 can be improved to avoid unnecessary computations in some cases.

Filtering Out Bounded Variables. The initialization of $\texttt{S}^{\texttt{val}}$ at line 32 can be only performed from unbound variables (and the last assigned variable), instead of the whole scope. We can maintain them in a reversible sparse set.

Last Modified Variable. It is not necessary to attempt to filter values out from the domain of a variable x if this was the only modified variable since the last call to enforceGAC(). Indeed, when updateTable() is executed, the new state of $\texttt{currTable}$ will be computed from $dom(x)$ or Δ_x only. Because every value of x had a support in $\texttt{currTable}$ the last time the propagator was called, we can omit filtering $dom(x)$ by initially removing x from $\texttt{S}^{\texttt{sup}}$.

7   Experiments

We experimented CT on $1,621$ CSP instances involving (positive) table constraints ($15\,\text{GB}$ of uncompressed files in format XCSP 2.1). This corresponds to a large variety of instances, taken from 37 series. For guiding search, we used binary branching with *domain over degree* as variable ordering heuristic and *min value* as value ordering heuristic. A timeout of $1,000\,\text{s}$ was used for each

instance. The tested GAC algorithms are CT, STR2 [17], STR3 [20], GAC4 [28,30], GAC4R [30], MDDR [30] and AC5TCRecomp [26]. All scripts, codes and benchmarks allowing to reproduce our experiments are available at https://bitbucket.org/pschaus/xp-table. The experiments were run on a 32-core machine (1400MHz cpu) with 100GB using Java(TM) SE Runtime Environment (build 1.8.0_60-b27) with 10GB of memory allocated (-Xmx option).

Performance Profiles. Let $t_{i,\,s}$ represent the time obtained with filtering algorithm $s \in S$ on instance $i \in I$. The performance ratio is defined as follows: $r_{i,\,s} = \frac{t_{i,\,s}}{\min\{t_{i,\,s}|s\in S\}}$. A ratio $r_{i,\,s} = 1$ means that s was the fastest on instance i. The performance profile [8] is a cumulative distribution function of the performance of s (speedup) compared to other algorithms: $\rho_s(\tau) = \frac{1}{|I|} \times |\{i \in I | r_{i,\,s} \le \tau\}|$.

Our results are visually aggregated to form a performance profile in Fig. 4 generated by means of the online tool [5] http://sites.uclouvain.be/performance-profile. Note that we filtered out the instances that (i) could not be solved within 1,000 s by all algorithms (ii) were solved in less than 2 s by the slowest algorithm, and (iii) required less than 500 backtracks. The final set of instances used to build the profile is composed of 227 instances. An interactive performance profile is also available at https://www.info.ucl.ac.be/~pschaus/assets/publi/performance-profile-ct to let the interested reader deactivate some family of instances to analyze the results more closely.
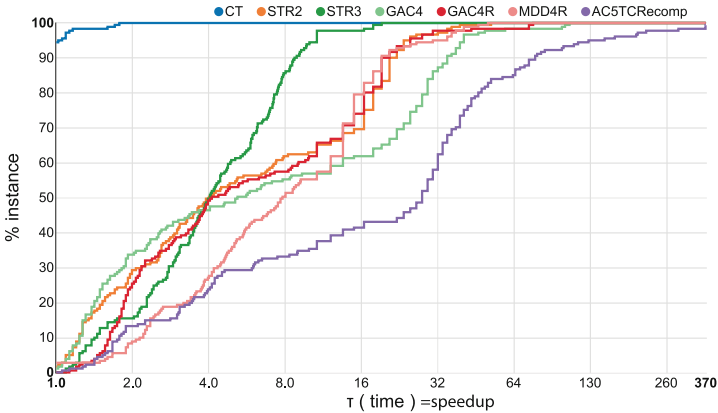
Fig. 4. Performance profile

Table 1 reports the speedup statistics of CT over the other algorithms. A first observation is that CT is the fastest algorithm on 94.47 % of the instances. Among all tested algorithms, AC5TCRecomp obtains the worse results. Then it is not clear which one among STR2, STR3, GAC4 and GAC4R is the second best algorithm. Based on the geometric mean speedup, STR3 seems to be the second best algorithm followed by STR2, GAC4R and MDD4R. Importantly,

Table 1. Speedup analysis of CT over the other algorithms. Column 'Best2' corresponds to a virtual second best solver (minimum time of all algorithms except CT).

Speedup	STR2	STR3	GAC4	GAC4R	MDD4R	AC5-TC	Best2
Geometric mean	5.09	4.03	7.05	6.15	6.57	19.22	2.75
Min	0.76	1.09	0.92	1.13	0.13	1.05	0.13
Max	88.58	51.04	173.24	208.52	50.84	1850.82	15.99
St. dev	10.64	4.36	19.67	18.57	9.46	134.13	2.87

one can observe that the geometric mean speedup of CT over the best of the other algorithms is about 2.75.

Impact of Resetting Operations. In Algorithm 2, the choice of being incremental or not, when updating `currTable`, depends on the size of several sets and is thus dynamic. We propose to analyze two variants of Algorithm 2 when this choice is static:

– Full incremental (CTI): only the body of the 'if' at line 10 is executed (deltas are systematically used).
– Full re-computation (CTR): only the body of the 'else' at line 14 is executed (domains are systematically used).

The performance profiles with these two variants are given in Fig. 5, and the speedup table of the static versions over the dynamic one is given in Table 2.
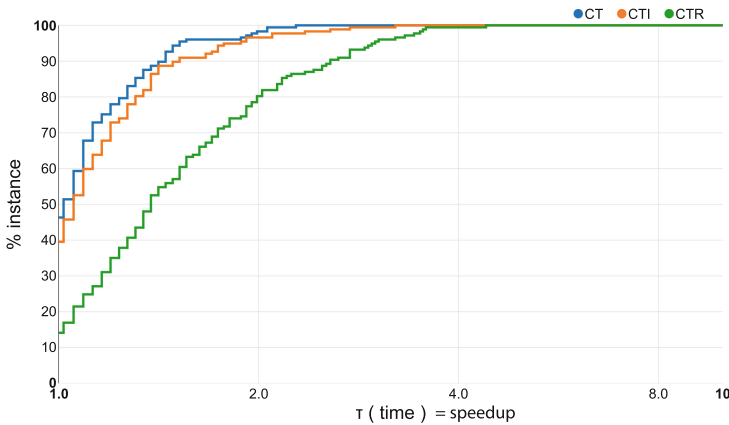

Fig. 5. Performance Profiles with dynamic (CT), recomputation (CTR) and incremental (CTI) strategies.

As can be seen from both the performance profiles and the speedup table, the dynamic version using the resetting operations dominates the static ones. The geometric mean speedup is around 4 % over CTI and 34 % over CTR.

Table 2. Speedup analysis of the two static variants over CT.

Speedup	CTI	CTR	Best
Geometric mean	1.04	1.34	0.99
Min	0.44	0.53	0.44
Max	3.23	4.39	1.96
St. dev	0.38	0.65	0.27

Contradiction with Previous Results. In [26], AC5TCRecomp was presented as being competitive with STR2. When we analyzed the code[4] of STR2 used in [26], it appeared that STR2 was implemented in Comet using built-in sets (triggering the garbage collection of Comet). We thus believe that the results and conclusions in [26] may over-penalize the performance of STR2. Our results also somehow contradict the results in [30] where STR3 and STR2 were dominated by MDD4R and GAC4R. When analyzing the performance of the implementation of STR2 and STR3 used in [30] with or-tools, it appears that it is not as competitive as that in AbsCon (sometimes slower by a factor of 3). The results presented in [30] may thus also over-penalize STR2 and STR3.

One additional contribution of this work is a fined-tuned implementation of the best filtering algorithms for table constraints. The implementation of these algorithms in OscaR was optimized, and checked to be close in performance to the ones by the original authors. For CT, STR2 and STR3, a comparison was made with AbsCon, and for CT, MDD4R and GAC4R, a comparison was made with or-tools. Our implementation required a development effort of 10 man-months in order to obtain an efficient implementation of each algorithm. It involved the expertise of several OscaR developers and a deep analysis of the existing implementations in AbsCon and or-tools. The implementation of all algorithms used in this paper is open-source and part of OscaR release 3.1.0.

8   Conclusion

In this paper, we have shown that Compact-Table (CT) is a robust algorithm that clearly dominates state-of-the-art propagators for table constraints. CT benefits from well-tried techniques: bitwise operations, residual supports, tabular reduction and resetting operations. We believe that CT can be easily implemented using the reversible sparse bit-set data structure.

References

1. Bessiere, C., Régin, J.-C.: Arc consistency for general constraint networks: preliminary results. In: Proceedings of IJCAI 1997, pp. 398–404 (1997)
2. Bessiere, C., Régin, J.-C., Yap, R., Zhang, Y.: An optimal coarse-grained arc consistency algorithm. Artif. Intell. **165**(2), 165–185 (2005)

---

[4] available at http://becool.info.ucl.ac.be.

3. Bliek, C.: Wordwise algorithms and improved heuristics for solving hard constraint satisfaction problems. Technical Report 12–96-R045, ERCIM (1996)
4. Briggs, P., Torczon, L.: An efficient representation for sparse sets. ACM Lett. Programm. Lang. Syst. **2**(1–4), 59–69 (1993)
5. Van Cauwelaert, S., Lombardi, M., Schaus, P.: A visual web tool to perform what-if analysis of optimization approaches. Technical report, UCLouvain (2016)
6. Cheng, K., Yap, R.: An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. Constraints **15**(2), 265–304 (2010)
7. Demeulenaere, J.: Efficient algorithms for table constraints. Technical report, Master Thesis, under the supervision of P. Schauss, UCLouvain (2015)
8. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
9. Floyd, R.W.: Nondeterministic algorithms. J. ACM **14**(4), 636–644 (1967)
10. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Data structures for generalised arc consistency for extensional constraints. In: Proceedings of AAAI 2007, pp. 191–197 (2007)
11. Gharbi, N., Hemery, F., Lecoutre, C., Roussel, O.: Sliced table constraints: combining compression and tabular reduction. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 120–135. Springer, Heidelberg (2014)
12. Van Hentenryck, P., Mairy, J.-B., Deville, Y.: Optimal and efficient filtering algorithms for table constraints. Constraints **19**(1), 77–120 (2014)
13. Jefferson, C., Nightingale, P.: Extending simple tabular reduction with short supports. In: Proceedings of IJCAI 2013, pp. 573–579 (2013)
14. Katsirelos, G., Walsh, T.: A compression algorithm for large arity extensional constraints. In: Proceedings of CP 2007, pp. 379–393 (2007)
15. Knuth, D.E.: The Art of Computer: Combinatorial Algorithms, vol. 4. Addison-Wesley (2015)
16. de Saint-Marcq, V.C., Schaus, P., Solnon, C., Lecoutre, C.: Sparse-sets for domain implementation. In: Proceeding of TRICS 2013, pp. 1–10 (2013)
17. Lecoutre, C.: STR2: Optimized simple tabular reduction for table constraints. Constraints **16**(4), 341–371 (2011)
18. Lecoutre, C., Boussemart, F., Hemery, F.: Exploiting multidirectionality in coarse-grained arc consistency algorithms. In: Proceedings of CP 2003, pp. 480–494 (2003)
19. Lecoutre, C., Hemery, F.: A study of residual supports in arc consistency. In: Proceedings of IJCAI 2007, pp. 125–130 (2007)
20. Lecoutre, C., Likitvivatanavong, C., Yap, R.: STR3: A path-optimal filtering algorithm for table constraints. Artif. Intell. **220**, 1–27 (2015)
21. Lecoutre, C., Szymanek, R.: Generalized arc consistency for positive table constraints. In: Proceedings of CP 2006, pp. 284–298 (2006)
22. Lecoutre, C., Vion, J.: Enforcing arc consistency using bitwise operations. Constraint Program. Lett. **2d**, 21–35 (2008)
23. Lhomme, O., Régin, J.-C.: A fast arc consistency algorithm for n-ary constraints. In: Proceedings of AAAI 2005, pp. 405–410 (2005)
24. Likitvivatanavong, C., Zhang, Y., Bowen, J., Freuder, E.C.: Arc consistency in MAC: a new perspective. In: Proceedings of CPAI 2004 Workshop held with CP 2004, pp. 93–107 (2004)
25. Mackworth, A.K.: Consistency in networks of relations. Artif. Intell. **8**(1), 99–118 (1977)
26. Mairy, J.-B., van Hentenryck, P., Deville, Y.: An optimal filtering algorithm for table constraints. In: Proceedings of CP 2012, pp. 496–511 (2012)

27. McGregor, J.J.: Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. Inf. Sci. **19**, 229–250 (1979)
28. Mohr, R., Masini, G.: Good old discrete relaxation. In: Proceedings of ECAI 1988, pp. 651–656 (1988)
29. Team, O.: OscaR: Scala in OR (2012). https://bitbucket.org/oscarlib/oscar
30. Perez, G., Régin, J.-C.: Improving GAC-4 for table and MDD constraints. In: Proceedings of CP 2014, pp. 606–621 (2014)
31. Srinivasan, A., Kam, T., Malik, S., Brayton, R.K.: Algorithms for discrete function manipulation. In: Proceedings of ICCAD 1990, pp. 92–95 (1990)
32. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM **23**(1), 31–42 (1976)
33. Ullmann, J.R.: Partition search for non-binary constraint satisfaction. Inf. Sci. **177**, 3639–3678 (2007)
34. Wang, R., Xia, W., Yap, R., Li, Z.: Optimizing simple table reduction with bitwise representation. In: Proceedings of IJCAI 2016 (2016)
35. Xia, W., Yap, R.: Optimizing STR algorithms with tuple compression. In: Proceedings of CP 2013, pp. 724–732 (2013)

Interval Constraints with Learning: Application to Air Traffic Control

Thibaut Feydy[1(✉)] and Peter J. Stuckey[1,2]

[1] Data61, CSIRO, Melbourne, Australia
{thibaut.feydy,peter.stuckey}@data61.csiro.au
[2] Department of Computing and Information Systems,
University of Melbourne, Melbourne, Australia

Abstract. Lazy Clause Generation (LCG) is a learning extension of Constraint Programming that combines the power of SAT and CP. In this paper we present an extension of Lazy Clause Generation from finite domain constraints to interval constraints, that is: non-linear constraints over the reals. Because LCG solvers must be able to negate literals involved in computation, LCG for intervals must represent both open and closed intervals. This makes LCG for intervals quite different from LCG for integers. We illustrate the advantage of the technology by solving a mixed integer non-linear Air Traffic Control problem .

1 Introduction

The capacities of European en-route Air Traffic Control (ATC) centers are far exceeded by a constant growth in air traffic demand, resulting in ever increasing flight delays. To overcome this issue, novel Air Traffic Management (ATM) schemes are designed while keeping the hard constraint of a minimal 5 nautical mile horizontal safety separation between every pair of aircraft. Nowadays, solutions to avoid conflicts are empirical, and human controllers rely on standard routes and traffic organization to devise them. However, the complexity of conflicts could grow tremendously within future ATM systems, should the aircraft fly on direct routes, from take-off airport to destination. Human controllers would no longer be able to solve them efficiently on their own, thus requiring automated solvers. Former approaches like [6] use local search (namely genetic algorithms) to solve the conflict problem. These meta-heuristics are well suited to solve large scale and difficult problems when no other relevant techniques are known, but stochastic search inherently lacks existence and optimality proofs. An interval constraint approach was offered in [8]. While the method allows proof of optimality and the existence of solutions, it does not scale to the size of a general air traffic sector. The difficulty in handling the required constraints is related to the fact that the separation must be kept at any time. In this paper we propose to solve this problem by extending an Interval Constraint Solver with Learning. Learning methods such as lazy clause generation [15] can exponentially reduce the search complexity and are particularly well suited to such a problem where some of the variables can be discretized. After presenting interval constraint

methods and its extensions to learning, we present the air traffic control models and their implementation and provide results validating the approach.

2   Preliminaries

2.1   Finite Domain Constraint Programming

A *valuation*, θ, is a mapping of variables to values, denoted $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. Define $vars(\theta) = \{x_1, \ldots, x_n\}$. A *primitive constraint*, c, is a set of valuations over a set of variables $vars(c)$. A valuation θ is a *solution* of c if $\{x \mapsto \theta(x) \mid x \in vars(c)\} \in c$. A *constraint* C is a conjunction of primitive constraints, which we often treat as a set. A valuation θ is a solution of constraint C if it is a solution for each $c \in C$. We write $C_1 \models C_2$ if every solution of C_1 is a solution of c_2.

An *atomic constraint* is a unary constraint of the form $\langle x = d \rangle$, $\langle x \neq d \rangle$, $\langle x \geq d \rangle$, $\langle x \leq d \rangle$, or *false*. We write atomic constraints in angle brackets to emphasize their special status. A *domain* D is a conjunction of atomic constraints. D is a *false domain* if it has no solutions. We use notation $D(x) = \{\theta(x) \mid \theta$ is a solution of $D\}$. A *singleton domain* is one where $|D(x)| = 1, x \in vars(D)$, and we let $\theta_D = \{x \mapsto d_x \mid x \in vars(D), D(x) = \{d_x\}\}$ in this case.

A *propagator* $p(c)$ for constraint c is an inference algorithm, it maps a domain D to a set of atomic constraints $p(c)(D)$, where $D \wedge c \models p(c)(D)$. We assume each propagator is *checking*, that is if $\forall x \in vars(c).|D(x)| = 1$ then $p(c)(D) = \emptyset$ if θ_D is a solution of c and $\{false\}$ otherwise. A propagation solver $prop(P, D)$ applied to a set of propagators P and a domain D repeatedly applies the propagators $p \in P$ until $p(D') = \emptyset$ for $p \in P$, and returns D'.

A *constraint satisfaction problem (CSP)* $P = (V, D, C)$ is a constraint C and domain constraint D over variables $V = vars(C) \cup vars(D)$. A CP solver solves the CSP by applying the propagation solver $prop(\{p(c) \mid c \in C\}, D)$ to obtain a new domain D', then if this is not a false domain or singleton domain, guessing an atomic constraint *decision* a, and solving the two problems $(V, D' \wedge a, C)$ and $(V, D' \wedge \neg a, C)$.

2.2   Lazy Clause Generation for Integers

Lazy clause generation (LCG) solvers [15] are hybrid CP and SAT solvers that combine CP propagation based solving with SAT nogood learning. An LCG solver represents an integer variable with initial domain $[l .. u]$ by the Boolean variables $[\![x = d]\!], l \leq d \leq u$ (*equality variables*) and $[\![x \geq d]\!], l < d \leq u$ (*bounds variables*). Note that each atomic constraint defined earlier, is exactly a Boolean literal using this representation: $\langle x = d \rangle$ is $[\![x = d]\!]$, $\langle x \neq d \rangle$ is $\neg[\![x = d]\!]$, $\langle x \geq d \rangle$ is $[\![x \geq d]\!]$ and $\langle x \leq d \rangle$ is $\neg[\![x \geq d + 1]\!]$.

The Boolean variables are connected to an integer domain propagator which ensures that they maintain a consistent representation of an integer variable, that is $[\![x \geq d + 1]\!] \rightarrow [\![x \geq d]\!], l < d < u$, and $[\![x = d]\!] \leftrightarrow [\![x \geq d]\!] \wedge \neg[\![x \geq d + 1]\!], l < d < u$, $[\![x = l]\!] \leftrightarrow \neg[\![x \geq l + 1]\!]$, and $[\![x = u]\!] \leftrightarrow [\![x \geq u]\!]$.

In LCG solvers propagators are also required to return explanations for each new consequence $l \in p(c)(D)$, that is an explanation clause $e \equiv l_1 \wedge \cdots l_n \to l$ where $\forall 1 \leq i \leq n, D \models l_i$ and $c \models e$. In LCG solvers during propagation [7,15], a *trail* of newly inferred literals representing atomic constraints is created, each of which has an explanation clause showing which previously true literals made it true.

When an LCG solver infers *false* it, like a SAT solver, repeatedly replaces literals in the explanation for the failure until only one literal that became true since the last decision remains. The resulting explanation of failure is the so called 1UIP nogood [14]. This nogood is then stored in the system as a new constraint (propagator), and the solver backjumps to the second last decision level in the nogood. At this point the nogood is guaranteed to propagate new information. See [15] for more details.

Example 1. Consider a CSP with constraints $x \geq y$, $t \geq 2 \to b$, $b \to x \leq 3z$, $b \to y \geq 2$, over integers x, y, z and t, and Boolean b and initial domain $D = \langle x \geq 0 \rangle \wedge \langle x \leq 10 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle y \leq 10 \rangle \wedge \langle z \geq 0 \rangle \wedge \langle z \leq 10 \rangle \wedge \langle t \geq 0 \rangle \wedge \langle t \leq 10 \rangle$. An initial decision $\langle z \leq 5 \rangle$ $(\neg [\![z \geq 6]\!])$ causes no propagation. The next decision $\langle t \geq 6 \rangle$ $([\![t \geq 6]\!])$ causes b which in turn causes $[\![y \geq 2]\!]$ and (with $\neg [\![z \geq 6]\!]$) $\neg [\![x \geq 2]\!]$, and these two propagate to *false*. The initial nogood is $[\![y \geq 2]\!] \wedge \neg [\![x \geq 2]\!] \to false$, replacing $\neg [\![x \geq 2]\!]$ by its reasons gives $\neg [\![z \geq 6]\!] \wedge b \wedge [\![y \geq 2]\!] \to false$, then replacing $[\![y \geq 2]\!]$ gives $\neg [\![z \geq 6]\!] \wedge b \to false$. The resulting 1UIP nogood is $[\![z \geq 6]\!] \vee \neg b$.   □

2.3   Interval Arithmetic

Given the discrete representation of numbers by computers it is impossible to solve continuous problems exactly. Interval constraint solvers use interval arithmetic [13] to compute sound approximations of the constraint system, through a combination of local consistencies and search.

Let $\mathbb{R}$ be the set of real numbers, and let $\mathbb{R}^\infty$ be $\mathbb{R} \cup \{+\infty, -\infty\}$. Let $\mathbb{F}$ be the subset of $\mathbb{R}$ of the representable floating-point numbers in a given format, and let $\mathbb{F}^\infty$ be $\mathbb{F} \cup \{+\infty, -\infty\}$. Let $\downarrow (r)$ (resp. $\uparrow (r)$) be the downward (resp. upward) roundings to $\mathbb{F}^\infty$ of a real number r. Given two numbers $a \in \mathbb{F} \cup \{-\infty\}$ and $b \in \mathbb{F} \cup \{+\infty\}$ the *closed interval* $[a, b]$ is the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$.

Less usual for interval arithmetic we will also consider open and semi-open intervals. The *open interval* (a, b) is the set $\{x \in \mathbb{R} \mid a < x < b\}$, while the two forms of *semi-open intervals* $(a, b]$ and $[a, b)$ represent the sets $\{x \in \mathbb{R} \mid a < x \leq b\}$ and $\{x \in \mathbb{R} \mid a \leq x < b\}$ respectively.

We will use $\mathbb{I}$ to represent the set of *closed intervals*, which is closed under intersection. We will use $\mathbb{I}^+$ to represent the set of (all) *intervals*, including open and semi-open intervals, which is also closed under intersection.

Given a closed interval I we define $\lfloor I \rfloor$ (resp. $\lceil I \rceil$) as the smallest (resp. largest) element of I.

Given a real operator $*$, the associated interval operator $\circledast$ is defined by $X \circledast Y = \bigcap_{\mathbb{I}} \{Z \mid \forall x \in X, \forall y \in Y, x * y \in Z\}$, e.g. $[a, b] \ominus [c, d] = [\downarrow (a - d), \uparrow (b - c)]$.

2.4   Interval Constraints Solving

A real (resp. interval) constraint is an atomic formula arising from a relation over real (resp. interval) expressions and variables. In practice interval constraint propagators enforce approximate consistencies, often *hull consistency* [3] or *box consistency* [2]. The original hull consistency algorithm $hc3$ decomposes constraints into primitives constraints implemented each by a corresponding propagator.

Example 2. The constraint c: $(x+y)+2*b = 0$ can be decomposed into $c_1 : z = x + y$ and $c_2 : z + 2 * b = 0$. The *hull consistent* propagator for the constraint c_1, with the domains X, Y, and Z computes the common fixed-point of the projection operators: $X \leftarrow X \cap (Z \ominus Y), Y \leftarrow Y \cap (Z \ominus X)$, and $Z \leftarrow Z \cap (X \oplus Y)$. $\square$

A refinement of the $hc3$ algorithm is $hc4$ [12] which avoids decomposing the constraints by working directly on a tree-like representation of constraints where each node is either a variable, a constant or a primitive function operator. The variables domains pruning is done through a forward evaluation of the tree followed by a backward top-down projection narrowing operation. During the top-down pruning the algorithm may prematurely end by the computation of an empty interval, in which case the constraint is inconsistent w.r.t the current domain.

Example 3. The constraint $c : (x + y) + (2 * b) = 0$ has the tree representation $c : (e_1 : (e_2 : x + y) + (e_3 : 2 * b)) = 0$. Given the domain X, Y, B, the evaluation phases computes $e_2.f = X \oplus Y$, $e_3.f = [2, 2] \otimes B$, $e_1.f = e_2.f \oplus e_3.f$. The top-down pruning phases enforces the projection $e_1.b \leftarrow e_1.f \cap [0, 0]$, $e_2.b \leftarrow e_2.f \cap (e_1.b \ominus e_3.f)$, $X \leftarrow X \cap (e_2.b \ominus Y)$, $Y \leftarrow Y \cap (e_2.b \ominus X)$, $e_3.b \leftarrow e_3.f \cap (e_1.b \ominus e_2.f)$, $B \leftarrow B \cap (e_3.b \oslash [2, 2])$. $\square$

3   Lazy Clause Generation for Intervals

The critical question in defining a learning solver is how to represent the changes in variables. A natural representation for interval variable x would be using atomic constraints of the form $\langle x \in I \rangle$, which record the entire interval I attached to the variable. Indeed there are finite domain learning solvers which take this approach [16]. The disadvantages of this approach is that resulting nogoods are unlikely to be very reusable, and the atomic constraints themselves interact in complex ways. A stronger disadvantage is that atomic constraints will be negated, and the negative form of these constraints is hard to reason about.

The obvious choice, analogous to the integer case is to use the atomic constraints $\langle x \geq a \rangle$, $\langle x \leq a \rangle$, $a \in \mathbb{F}$. This allows us to represent all closed intervals. Unlike the integer case we cannot get away with a single set of bounds variables since $\neg \langle x \geq a \rangle \not\leftrightarrow \langle x \leq a \rangle$. Hence we need 2 sets of Boolean variables $[\![x \geq a]\!]$ and $[\![x \leq a]\!]$. Since $\langle x < a \rangle \leftrightarrow \neg[\![x \geq a]\!]$ and $\langle x > a \rangle \leftrightarrow \neg[\![x \leq a]\!]$, we will be able to represent open and semi-open intervals.

Clearly we cannot create a Boolean variable for each possible atomic constraint $\langle x \geq a \rangle$, $\langle x \leq a \rangle$, $a \in \mathbb{F}$ for variable x *apriori*, there are far too many. Indeed even during propagation far too many atomic constraints will appear for us to represent them each by a Boolean variable. In an LCG (and SAT) solver each Boolean variable is a non-trivial data structure storing watch lists, activity counts, and any associated atomic constraint.

To avoid the cost of creating many Boolean variables during propagation we make use of a *stateless* atomic constraint representation (tagged pointer), which carries its meaning with it, and use this for propagation, and recording the implication graph in the trail, and the explanations of propagation, and for building explanations. Most atomic constraints will appear on the trail, and simply be removed by backtracking/backjumping. We will only create Boolean variables corresponding to atomic constraints that end up in the nogoods that are created.

Example 4. Reconsider the CSP of Example 1 where now x, y, z and t are interval variables. An initial decision $\langle z \leq 5 \rangle$ causes no propagation. The next decision $\langle t \geq 6 \rangle$ causes b which in turn causes $\langle y \geq 2 \rangle$ and (with $\langle z \leq 5 \rangle$) $\langle x \leq \uparrow \frac{5}{3} \rangle$, and these two propagate to *false*. The initial nogood is $\langle y \geq 2 \rangle \wedge \langle x \leq \uparrow \frac{5}{3} \rangle \rightarrow \textit{false}$, replacing $\langle x \leq \uparrow \frac{5}{3} \rangle$ by its reasons gives $\langle z \leq 5 \rangle \wedge b \wedge \langle y \geq 2 \rangle \rightarrow \textit{false}$, then replacing $\langle y \geq 2 \rangle$ gives $\langle z \leq 5 \rangle \wedge b \rightarrow \textit{false}$. The resulting 1UIP nogood is $\neg \llbracket z \leq 5 \rrbracket \vee \neg b$. Note how the entire process uses atomic constraints, except the final stored nogood which uses literals. $\qquad\square$

A critical component of the interval learning solver is the *interval domain propagator* which is responsible for mapping interval domain information to atomic constraints and any associated Boolean literals, and vice versa.

The domain of interval variable x is implemented as a sorted map from float values a to atomic constraints $\langle x < a \rangle$, $\langle x \leq a \rangle$, $\langle x \geq a \rangle$, and $\langle x > a \rangle$. We cache the current upper and lower bounds for x, but not their positions in the map. Changes to $D(x)$ require walking the map to determine which atomic constraints become *true* or *false*. Note that in this way the domain propagator for x also maintains the consistency of the Boolean literals associated with x, which will be added to the queue for propagation.

When a new atomic constraint is created, it is inserted appropriately in the map. Note usually a new atomic constraint is only created by propagation which makes it true, so we can implement this simply by walking the map from the current bound to the position of the new bound and inserting it, since we have to walk the map setting the other atomic constraints in the path *true* or *false* appropriately.

3.1   Propagation with Learning

Note that although we must represent open, semi-open and closed intervals, in order to have the representation of intervals closed under negation, the interval propagation almost always relaxes intervals to be closed. The only cases where

this does not occur is when no floating point operations occur on the interval bounds, for example in equality, min and max. Clearly the resulting computation is still safe.

In order to provide explanations for variable domain updates, interval operations are *augmented* to maintain the reasons for their results, in the form of a set of atoms per bound. For example the augmentation $\boxplus$ of the operator $\oplus$ is defined as $(X, l_x, u_x) \boxplus (Y, l_y, u_y) = (X \oplus Y, l_x \cup l_y, u_x \cup u_y)$. Given a variable x with a domain X let $\Delta(x) = (X, \{\langle x \geq \lfloor X \rfloor\rangle\}, \{\langle x \leq \lceil X \rceil\rangle\})$. These augmented operators are used in the implementation of propagators, to derive reasons for failure or variables bounds updates,

Example 5. Reconsiding Example 3 in the context of learning, the bottom-up evaluation now computes $e_2.f = \Delta(x) \boxplus \Delta(y)$, $e_3.f = ([2, 2], \{\}, \{\}) \boxtimes \Delta(b)$, $e_1.f = e_2.f \boxplus e_3.f$. The top-down pruning phases enforces the projection $e_1.b \leftarrow e_1.f \cap ([0, 0], \{\}, \{\})$, $e_2.b \leftarrow e_2.f \cap (e_1.b \boxminus e_3.f)$, $e_3.b \leftarrow e_3.f \cap (e_1.b \boxminus e_2.f)$ and the following potential updates augmented with explanations for x, y, and b : $\Delta(x)\boxdot(e_2.b \boxminus \Delta(y))$, $\Delta(y)\boxdot(e_2.b \boxminus \Delta(x))$ and $\Delta(b)\boxdot(e_3.b \boxslash [2,2])$.

Consider the constraint with domains $x \in [-2, 0]$, $y \in [-1, 0]$ and $b \in [0, 1]$ when b changes to $[1,1]$, ignoring any rounding problems for simplicity. We recalculate $e_3.f = ([2, 2], \{\langle b \geq 1\rangle\}, \{\})$, $e_2.b = ([-2, -2], \{\}, \{\langle b \geq 1\rangle\})$, $e_2.b \boxminus \Delta(y) = ([-2, -1], \{\langle y \leq 0\rangle\}, \{\langle b \geq 1\rangle, \langle y \geq -1\rangle\})$, $x = ([-2, -1], \{\langle x \geq -2\rangle\}, \{\langle b \geq 1\rangle, \langle y \geq -1\rangle\})$. The explanation for the change in x is $\langle b \geq 1\rangle \wedge \langle y \geq -1\rangle \rightarrow \langle x \leq -1\rangle$. $\qquad\square$

In practice it is possible, during forward evaluation, to simply flag bits indicating which of an expression children bounds are used during evaluation of its f field to avoid the systematic creation and union of sets of atoms. A reason will be then reconstructed, if needed, when a variable bound is updated.

4   Mixed Models

In this section we present the models first introduced in [8]. An aircraft i is characterized by an initial position $p_i(0) = (x_i(0), y_i(0))$, a speed v_i, a heading θ_i and a waypoint or destination w_i along its path (see Fig. 1). We consider horizontal maneuvers between aircraft at the same altitude. At any given time, two aircraft are in conflict when the distance between them is less than a safety separation d. The considered maneuvers for maintaining separation involve deviations of the aircraft headings. Given that these maneuvers are orders for pilots, the starting time and deviation angle of a maneuver are discrete variables, indeed arbitrarily precise orders would be unrealistic.

4.1   Horizontal TCAS Model

This simple model is for emergency situations and could be used for a real-time Traffic Collision Avoidance System (TCAS): at the initial time, deviations are
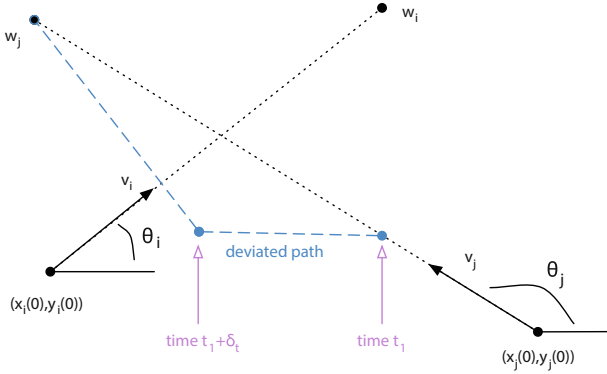
Fig. 1. Illustration of a deviated path to avoid conflict in the human controller model.

applied to the aircraft headings to avoid conflicts. It has one discrete decision variable α_i per aircraft i.

Given two aircraft i and j, let $\boldsymbol{v}_{ij}$ be the relative speed and $\boldsymbol{p}_{ij}(t)$ the relative position between them at time t, and d the safety distance. We have $\boldsymbol{p}_{ij}(t) = \boldsymbol{p}_{ij}(0) + \boldsymbol{v}_{ij}(t - 0)$. These two aircraft are not in conflict at a given time t if the distance separating them is greater than d: $P(i,j) = \boldsymbol{p}_{ij}(t)^2 - d^2 > 0$. If the discriminant $\Delta_{P(i,j)}$ of $P(i,j)$ is negative, these two aircraft will not enter into conflict, hence the inequality constraint per pair of aircraft is:

$$(\boldsymbol{p}_{ij}(0)\boldsymbol{v}_{ij})^2 - (\boldsymbol{p}_{ij}(0)^2 - d^2)v_{ij}^2 < 0$$

with :

$$\boldsymbol{p}_{ij}(0) = \begin{pmatrix} x_i(0) - x_j(0) \\ y_i(0) - y_j(0) \end{pmatrix} \qquad \boldsymbol{v}_{ij} = \begin{pmatrix} v_i cos(\theta_i + \alpha_i) - v_j cos(\theta_j + \alpha_j) \\ v_i sin(\theta_i + \alpha_i) - v_j sin(\theta_j + \alpha_j) \end{pmatrix}$$

4.2   Horizontal Human Controller Model

In this model we consider that the aircraft is initially heading toward a waypoint. To avoid a conflict, it is possible to deviate the aircraft from its original heading, at some time t_1. After an amount of time δ_t it will then head back toward its original destination. The path of an aircraft p is then composed of three segments s_{p1}, s_{p2} and s_{p3}. Given a pair of aircraft i and j, a conflict can arise for each pair of segments (s_{ix}, s_{jy}), resulting in 9 avoidance constraints per pair of aircraft.

Given two segments s_{ix}, s_{jy}, let $P(s_{ix}, s_{jy})$ be the associated distance polynomial as defined in the previous model. The avoidance constraint is defined by the following disjunction :

– there is no common time segment during which the aircraft i flies over s_{ix} and aircraft j flies over s_{jy}, or
– the discriminant of $P(s_{ix}, s_{jy})$ is negative, or
– the roots of $P(s_{ix}, s_{jy})$ are outside the common flight time.

5   Experiments

The lazy clause generation solver *Chuffed* [5] was extended with interval constraint support, which can be used both with or without learning. The optimization strategy chosen was the minimisation of the sum of the absolute deviations $|\alpha_i|$ which is a good approximation of how disruptive the solution is.

Other possible optimization strategies would be the minimization of number of deviated aircraft, which is a relevant criterion for a human controller, or to minimize the total lengthening of the paths, which better captures the airline operators' concerns.

In Table 1, we compare the performance of Interval Constraints without learning (IC) and with Lazy Clause Generation for simple avoidance problems (tcas$x$) and human controller model (hmc$x$) involving 4, 8 and 12 aircraft with a 90 s timeout. We obtain an exponential search space reduction from learning, with IC only solving the smaller human controller model problem. Since the aircrafts are constrained pairwise, it is likely that the nogoods transpose well to different parts of the search space. The benchmarks are available in MiniZinc format at people.unimelb.edu.au/pstuckey/atc.

Table 1. Comparison of Interval Constraints with and without learning.

Problem	IC		LCG			Problem	IC		LCG		
	#bts	t(s)	#lits	#bts	t(s)		#bts	t(s)	#lits	#bts	t(s)
tcas4	13	0.1	65	26	0.1	hcm4	513342	60.1	21206	27621	4.4
tcas8	1001	0.2	237	128	0.1	hcm8	—	>90	28521	42372	21.2
tcas12	52863	4.65	2107	1655	0.3	hcm12	—	>90	43234	64890	62.1

6   Related Work and Conclusion

Constraint systems such as ECL^iPS^e [4] support both integers and interval constraints. The framework presented in [11] and the SMT solver HySAT [9], based on the iSAT algorithm [10], combine interval constraint propagation with the learning framework of SMT to solve real constraints, implementing a form of $hc3$ augmented by explanations (as opposed to $hc4$ that we implement).

The SMT approaches do not tightly integrate the handling of integer and interval variables which is a distinct disadvantage for applications such as ATC. They both elide the issue of too many literals appearing in the trail, which may be because the benchmarks they use are quite distinct from those appearing in typical CP interval problems where interval propagation can take many iterations to quiesce. Hence it appears the implementation issues for LCG and SMT for intervals are quite different.

The domain of Air Traffic Management is very complex and contains many hard combinatorial problems. Although there is little existing work regarding

continuous or mixed problems, CP approaches has been developed for many of the combinatorial problems in this area such as arrival management, runway allocation, workload management. See [1] for a survey.

References

1. Allignol, C., Barnier, N., Flener, P., Pearson, J.: Constraint programming for air traffic management: a survey. Knowl. Eng. Rev. **27**(03), 361–392 (2012)
2. Benhamou, F., McAllester, D., Van Hentenryck, P.: Clp (intervals) revisited. Rapport technique, p. 30. Citeseer (1994)
3. Benhamou, F.: Interval constraint logic programming. In: Podelski, A. (ed.) Constraint Programming: Basics and Trends. LNCS, vol. 910, pp. 1–21. Springer, Heidelberg (1995)
4. Brisset, P., Sakkout, H.E., Fruhwirth, T., Gervet, C., Harvey, W., Meier, M., Novello, S., Le Provost, T., Schimpf, J., Shen, K., Wallace, M.: ECLiPSe Constraint Library Manual, October 2005
5. Chu, G.G.: Improving combinatorial optimization. Ph.d. thesis, The University of Melbourne (2011)
6. Durand, N., Alliot, J.M., Noailles, J.: Automatic aircraft conflict resolution using genetic algorithms. In: Proceedings of the 1996 ACM Symposium on Applied Computing, pp. 289–298. ACM (1996)
7. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 352–366. Springer, Heidelberg (2009)
8. Feydy, T., Barnier, N., Brisset, P., Durand, N.: Mixed conflict model for air traffic control. In: IntCp 2005, Workshop on Interval analysis, constraint propagation, applications (2005)
9. Fränzle, M., Herde, C.: Hysat: an efficient proof engine for bounded model checking of hybrid systems. Formal Methods Syst. Des. **30**(3), 179–198 (2007)
10. Franzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. J. Satisfiability, Boolean Model. Comput. **1**, 209–236 (2007)
11. Haller, L., Griggio, A., Brain, M., Kroening, D.: Deciding floating-point logic with systematic abstraction. In: Formal Methods in Computer-Aided Design (FMCAD 2012), pp. 131–140. IEEE (2012)
12. Ilog, S.: Revising hull and box consistency. In: Logic Programming: Proceedings of the 1999 International Conference on Logic Programming, p. 230. MIT press (1999)
13. Moore, R.E.: Interval Analysis, vol. 4. Prentice-Hall, Englewood Cliffs (1966)
14. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 39th Design Automation Conference (DAC 2001) (2001)
15. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
16. Veksler, M., Strichman, O.: Learning general constraints in CSP. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 410–426. Springer, Heidelberg (2015). http://dx.doi.org/10.1007/978-3-319-18008-3_28

Backdoors to Tractable Valued CSP

Robert Ganian, M.S. Ramanujan, and Stefan Szeider[(✉)]

Algorithms and Complexity Group, TU Wien, Vienna, Austria
`rganian@gmail.com`, `ramanujan@tuwien.ac.at`, `stefan@szeider.net`

Abstract. We extend the notion of a strong backdoor from the CSP setting to the Valued CSP setting (VCSP, for short). This provides a means for augmenting a class of tractable VCSP instances to instances that are outside the class but of small distance to the class, where the distance is measured in terms of the size of a smallest backdoor. We establish that VCSP is fixed-parameter tractable when parameterized by the size of a smallest backdoor into every tractable class of VCSP instances characterized by a (possibly infinite) tractable valued constraint language of finite arity and finite domain. We further extend this fixed-parameter tractability result to so-called "scattered classes" of VCSP instances where each connected component may belong to a different tractable class.

1 Introduction

Valued CSP (or VCSP for short) is a powerful framework that entails among others the problems CSP and MAX-CSP as special cases [26]. A VCSP instance consists of a finite set of cost functions over a finite set of variables which range over a domain D, and the task is to find an instantiation of these variables that minimizes the sum of the cost functions. The VCSP framework is robust and has been studied in different contexts in computer science. In its full generality, VCSP considers cost functions that can take as values the rational numbers and positive infinity. CSP (feasibility) and Max-CSP (optimisation) arise as special cases by limiting the values of cost functions to $\{0, \infty\}$ and $\{0, 1\}$, respectively. Clearly VCSP is in general intractable. Over the last decades much research has been devoted into the identification of tractable VCSP subproblems. An important line of this research (see, e.g., [17,18,25]) is the characterization of tractable VCSPs in terms of restrictions on the underlying *valued constraint language Γ*, i.e., a set Γ of cost functions that guarantees polynomial-time solvability of all VCSP instances that use only cost functions from Γ. The VCSP restricted to instances with cost functions from Γ is denoted by VCSP[Γ].

In this paper we provide algorithmic results which allow us to gradually augment a tractable VCSP based on the notion of a (strong) *backdoor* into a tractable class of instances, called the *base class*. Backdoors where introduced by Williams *et al.* [27,28] for SAT and CSP and generalize in a natural way to VCSP. Let $\mathcal{C}$ denote a tractable class of VCSP instances over a finite domain D.

A backdoor of a VCSP instance $\mathcal{P}$ into $\mathcal{C}$ is a (small) subset B of the variables of $\mathcal{P}$ such that for all partial assignments α that instantiate B, the restricted instance $\mathcal{P}|_\alpha$ belongs to the tractable class $\mathcal{C}$. Once we know such a backdoor B of size k we can solve $\mathcal{P}$ by solving at most $|D|^k$ tractable instances. In other words, VCSP is then *fixed parameter tractable* parameterized by backdoor size. This is highly desirable as it allows us to scale the tractability for $\mathcal{C}$ to instances outside the class, paying for an increased "distance" from $\mathcal{C}$ only by a larger constant factor.

In order to apply this backdoor approach to solving a VCSP instance, we first need to *find* a small backdoor. This turns out to be an algorithmically challenging task. The fixed-parameter tractability of backdoor detection has been subject of intensive research in the context of SAT (see, e.g., [16]) and CSP (see, e.g., [2]). In this paper we extend this line of research to VCSP.

First we obtain some basic and fundamental results on backdoor detection when the base class is defined by a valued constraint language Γ. We obtain fixed-parameter tractability for the detection of backdoors into VCSP[Γ] where Γ is a valued constraint language with cost functions of bounded arity. In fact, we show the stronger result: fixed-parameter tractability also holds with respect to *heterogeneous* base classes of the form VCSP[Γ_1] $\cup \cdots \cup$ VCSP[Γ_ℓ] where different assignments to the backdoor variables may result in instances that belong to different base classes VCSP[Γ_i]. A similar result holds for CSP but the VCSP setting is slightly more complicated as a valued constraint language of finite arity over a finite domain is not necessarily finite.

Secondly, we extend the basic fixed-parameter tractability result to so-called *scattered* base classes of the form VCSP[Γ_1] $\oplus \cdots \oplus$ VCSP[Γ_ℓ] which contain VCSP instances where each connected component belongs to a tractable class VCSP[Γ_i] for some $1 \leq i \leq \ell$—again in the heterogeneous sense that for different assignments to the backdoor variables a single component of the reduced instance may belong to different classes VCSP[Γ_i]. Backdoors into a scattered base class can be much smaller than backdoors into each single class it is composed of, hence the gain is huge if we can handle scattered classes. This boost in scalability does not come for free. Indeed, already the "crisp" case of CSP, which was the topic of a recent SODA paper [14], requires a sophisticated algorithm which makes use of advanced techniques from parameterized algorithm design. This algorithm works under the requirement that the constraint languages contain all unary constraints (i.e., is conservative); this is a reasonable requirement as one needs these unary cost functions to express partial assignments (see also Sect. 2 for further discussion). Here we lift the crisp case to general VCSP, and this also represents our main technical contribution.

To achieve this, we proceed in two phases. First we transform the backdoor detection problem from a general scattered class VCSP(Γ_1) $\oplus \cdots \oplus$ VCSP(Γ_ℓ) to a scattered class VCSP(Γ_1') $\oplus \cdots \oplus$ VCSP(Γ_ℓ') over *finite* valued constraint languages Γ_i'. In the subsequent second phase we transform the problem to a backdoor detection problem into a scattered class VCSP(Γ_1'') $\oplus \cdots \oplus$ VCSP(Γ_ℓ'') where each Γ_i'' is a finite crisp language; i.e., we reduce from the VCSP setting to

the CSP setting. We believe that this sheds light on an interesting link between backdoors in the VCSP and CSP settings. The latter problem can now be solved using the known algorithm [14].

Related Work

Williams *et al.* [27,28] introduced backdoors for CSP or SAT as a theoretical tool to capture the overall combinatorics of instances. The purpose was an analysis of the empirical behaviour of backtrack search algorithms. Nishimura et al. [22] started the investigation on the parameterized complexity of *finding* a small SAT backdoor and using it to solve the instance. This lead to a number of follow-up work (see [16]). Parameterized complexity provides here an appealing framework, as given a CSP instance with n variables, one can trivially find a backdoor of size $\leq k$ into a fixed tractable class of instances by trying all subsets of the variable set containing $\leq k$ variables; but there are $\Theta(n^k)$ such sets, and therefore the running time of this brute-force algorithm scales very poorly in k. Fixed-parameter tractability removes k from the exponent providing running times of the form $f(k)n^c$ which yields a significantly better scalability in backdoor size.

Extensions to the basic notion of a backdoor have been proposed, including backdoors with empty clause detection [6], backdoors in the context of learning [7], heterogeneous backdoors where different instantiations of the backdoor variables may result in instances that belong to different base classes [15], and backdoors into scattered classes where each connected component of an instance may belong to a different tractable class [14]. Le Bras *et al.* [20] used backdoors to speed-up the solution of hard problems in materials discovery, using a crowd sourcing approach to find small backdoors.

The research on the parameterized complexity of backdoor detection was also successfully extended to other problem areas including disjunctive answer set programming [10,11], abstract argumentation [9], and integer linear programming [13]. There are also several papers that investigate the parameterized complexity of backdoor detection for CSP. Bessière *et al.* [1], considered "partition backdoors" which are sets of variables whose deletion partitions the CSP instance into two parts, one falls into a tractable class defined by a conservative polymorphism, and the other part is a collection of independent constraints. They also performed an empirical evaluation of the backdoor approach which resulted in promising results. Gaspers *et al.* [15] considered heterogeneous backdoors into tractable CSP classes that are characterized by polymorphisms. A similar approach was also undertaken by Carbonnel *et al.* [3] who also considered base classes that are "h-Helly" for a fixed integer h under the additional assumption that the domain is a finite subset of the natural numbers and comes with a fixed ordering.

2   Preliminaries

2.1   Valued Constraint Satisfaction

For a tuple t, we shall denote by $t[i]$ its i-th component. We shall denote by $\mathcal{Q}$ the set of all rational numbers, by $\mathcal{Q}_{\geq 0}$ the set of all nonnegative rational numbers, and by $\overline{\mathcal{Q}}_{\geq 0}$ the set of all nonnegative rational numbers together with positive infinity, ∞. We define $\alpha + \infty = \infty + \alpha = \infty$ for all $\alpha \in \overline{\mathcal{Q}}_{\geq 0}$, and $\alpha \cdot \infty = \infty$ for all $\alpha \in \mathcal{Q}_{\geq 0}$. The elements of $\overline{\mathcal{Q}}_{\geq 0}$ are called *costs*.

For every fixed set D and $m \geq 0$, a function φ from D^m to $\overline{\mathcal{Q}}_{\geq 0}$ will be called a *cost function* on D of arity m. D is called the *domain*, and here we will only deal with finite domains. If the range of φ is $\{0, \infty\}$, then φ is called a *crisp* cost function.

With every relation R on D, we can associate a crisp cost function φ_R on D which maps tuples in R to 0 and tuples not in R to ∞. On the other hand, with every m-ary cost function φ we can associate a relation R_φ defined by $(x_1, \ldots, x_m) \in R_\varphi \Leftrightarrow \varphi(x_1, \ldots, x_m) < \infty$. In the view of the close correspondence between crisp cost functions and relations we shall use these terms interchangeably in the rest of the paper.

A VCSP instance consists of a set of variables, a set of possible values, and a multiset of valued constraints. Each valued constraint has an associated cost function which assigns a cost to every possible tuple of values for the variables in the scope of the valued constraint. The goal is to find an assignment of values to all of the variables that has the minimum total cost. A formal definition is provided below.

Definition 1 (VCSP). *An instance $\mathcal{P}$ of the* VALUED CONSTRAINT SATIS-FACTION PROBLEM, *or* VCSP, *is a triple $(V, D, \mathcal{C})$ where V is a finite set of* variables, *which are to be assigned values from the set D, and $\mathcal{C}$ is a multiset of* valued constraints. *Each $c \in \mathcal{C}$ is a pair $c = (\boldsymbol{x}, \varphi)$, where $\boldsymbol{x}$ is a tuple of variables of length m called the* scope *of c, and $\varphi : D^m \to \overline{\mathcal{Q}}_{\geq 0}$ is an m-ary cost function. An* assignment *for the instance $\mathcal{P}$ is a mapping τ from V to D. We extend τ to a mapping from V^k to D^k on tuples of variables by applying τ componentwise. The* cost *of an assignment τ is defined as follows:*

$$Cost_{\mathcal{P}}(\tau) = \sum_{(\boldsymbol{x}, \varphi) \in \mathcal{C}} \varphi(\tau(\boldsymbol{x})).$$

The task for VCSP is the computation of an assignment with minimum cost, called a solution *to $\mathcal{P}$.*

For a constraint c, we will use $\mathbf{var}(c)$ to denote the set of variables which occur in the scope of c. We will later also deal with the *constraint satisfaction problem*, or CSP. Having already defined VCSP, it is advantageous to simply define CSP as the special case of VCSP where each valued constraint has a crisp cost function.

The following representation of a cost function will sometimes be useful for our purposes. A *cost table* for an m-ary cost function φ is a table with D^m rows

and $m+1$ columns with the following property: each row corresponds to a unique tuple $\boldsymbol{a} = (a_1, \ldots, a_m) \in D^m$, for each $i \in [m]$ the position i of this row contains a_i, and position $m + 1$ of this row contains $\varphi(a_1, \ldots, a_m)$.

A *partial assignment* is a mapping from $V' \subseteq V$ to D. Given a partial assignment τ, the *application* of τ on a valued constraint $c = (\boldsymbol{x}, \varphi)$ results in a new valued constraint $c|_\tau = (\boldsymbol{x}', \varphi')$ defined as follows. Let $\boldsymbol{x}' = \boldsymbol{x} \setminus V'$ (i.e., $\boldsymbol{x}'$ is obtained by removing all elements in $V \cap \boldsymbol{x}$ from $\boldsymbol{x}$) and $m' = |\boldsymbol{x}'|$. Then for each $\boldsymbol{a}' \in D^{m'}$, we set $\varphi'(\boldsymbol{a}') = \varphi(\boldsymbol{a})$ where for each $i \in [m]$

$$\boldsymbol{a}[i] = \begin{cases} \tau(\boldsymbol{x}[i]) & \text{if } \boldsymbol{x}[i] \in V' \\ \boldsymbol{a}'[i - j] & \text{otherwise, where } j = |\{ \boldsymbol{x}[p] \mid p \in [i] \} \cap V'|. \end{cases}$$

Intuitively, the tuple $\boldsymbol{a}$ defined above is obtained by taking the original tuple $\boldsymbol{a}'$ and enriching it by the values of the assignment τ applied on the "missing" variables from $\boldsymbol{x}$. In the special case when $\boldsymbol{x}'$ is empty, the valued constraint $c|_\tau$ becomes a nullary constraint whose cost function φ' will effectively be a constant. The application of τ on a VCSP instance $\mathcal{P}$ then results in a new VCSP instance $\mathcal{P}|_\tau = (V \setminus V', D, \mathcal{C}')$ where $\mathcal{C}' = \{ c|_\tau \mid c \in \mathcal{C} \}$. It will be useful to observe that applying a partial assignment τ can be done in time linear in $|\mathcal{P}|$ (each valued constraint can be processed independently, and the processing of each such valued constraint consists of merely pruning the cost table).

2.2 Valued Constraint Languages

A *valued constraint language* (or *language* for short) is a set of cost functions. The arity of a language Γ is the maximum arity of a cost function in Γ, or ∞ if Γ contains cost functions of arbitrarily large arities. Each language Γ defines a set $\mathrm{VCSP}[\Gamma]$ of VCSP instances which only use cost functions from Γ; formally, $(V, D, \mathcal{C}) \in \mathrm{VCSP}[\Gamma]$ iff each $(\boldsymbol{x}, \varphi) \in \mathcal{C}$ satisfies $\varphi \in \Gamma$. A language is *crisp* if it contains only crisp cost functions.

A language Γ is *globally tractable* if there exists a polynomial-time algorithm which solves $\mathrm{VCSP}[\Gamma]$.[1] Similarly, a class $\mathcal{H}$ of VCSP instances is called tractable if there exists a polynomial-time algorithm which solves $\mathcal{H}$. For technical reasons, we will implicitly assume that every language contains all nullary cost functions (i.e., constants); it is easily seen that adding such cost functions into a language has no impact on its tractability.

There are a few other properties of languages that will be required to formally state our results. A language Γ is *efficiently recognizable* if there exists a polynomial-time algorithm which takes as input a cost function φ and decides whether $\varphi \in \Gamma$. We note that every finite language is efficiently recognizable.

A language Γ is *closed under partial assignments* if for every instance $\mathcal{P} \in \mathrm{VCSP}[\Gamma]$ and every partial assignment τ on $\mathcal{P}$ and every valued constraint $c = (\boldsymbol{x}, \varphi)$ in $\mathcal{P}$, the valued constraint $c|_\tau = (\boldsymbol{x}', \varphi')$ satisfies $\varphi' \in \Gamma$. The *closure of*

---

[1] The literature also defines the notion of *tractability* [17,19], which we do not consider here. We remark that, to the best of our knowledge, all known tractable constraint languages are also globally tractable [17,19].

a language Γ under partial assignments, is the language $\Gamma' \supseteq \Gamma$ containing all cost functions that can be obtained from Γ via partial assignments; formally, Γ' contains a cost function φ' if and only if there exists a cost function $\varphi \in \Gamma$ such that for a constraint $c = (\boldsymbol{x}, \varphi)$ and an assignment $\tau : X \to D$ defined on a subset $X \subseteq \mathbf{var}(c)$ we have $c|_\tau = (\boldsymbol{x}', \varphi')$.

If a language Γ is closed under partial assignments, then also VCSP[Γ] is closed under partial assignments, which is a natural property and provides a certain robustness of the class. This robustness is also useful when considering backdoors into VCSP[Γ] (see Sect. 3), as then every superset of a backdoor remains a backdoor. Incidentally, being closed under partial assignments is also a property of tractable classes defined in terms of a polynomial-time subsolver [27,28] where the property is called *self-reducibility*.

A language is *conservative* if it contains all unary cost functions [18]. We note that being closed under partial assignments is closely related to the well-studied property of conservativeness. Crucially, for every conservative globally tractable language Γ, its closure under partial assignments Γ' will also be globally tractable; indeed, one can observe that every instance $\mathcal{P} \in$ VCSP[Γ'] can be converted, in linear time, to a solution-equivalent instance $\mathcal{P}' \in$ VCSP[Γ] by using infinity-valued (or even sufficiently high-valued) unary cost functions to model the effects of partial assignments.

2.3 Parameterized Complexity

We give a brief and rather informal review of the most important concepts of parameterized complexity. For an in-depth treatment of the subject we refer the reader to other sources [5,8,12,21].

The instances of a parameterized problem can be considered as pairs (I, k) where I is the *main part* of the instance and k is the *parameter* of the instance; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* (FPT) if instances (I, k) of size n (with respect to some reasonable encoding) can be solved in time $\mathcal{O}(f(k)n^c)$ where f is a computable function and c is a constant independent of k. The function f is called the *parameter dependence*, and algorithms with running time in this form are called *fixed-parameter algorithms*. Since the parameter dependence is usually super-polynomial, we will often give the running times of our algorithms in $\mathcal{O}^*$ notation which suppresses polynomial factors. Hence the running time of an FPT algorithm can be simply stated as $\mathcal{O}^*(f(k))$.

The exists a completeness theory which allows to obtain strong theoretical evidence that a parameterized problem is *not* fixed-parameter tractable. This theory is based on a hierarchy of parameterized complexity classes W[1] $\subseteq$ W[2] $\subseteq$ ... where all inclusions are believed to be proper. If a parameterized problem is shown to be W[i]-hard for some $i \geq 1$, then the problem is unlikely to be fixed-parameter tractable, similarly to an NP-complete problem being solvable in polynomial time [5,8,12,21].

3   Backdoors into Tractable Languages

This section is devoted to establishing the first general results for finding and exploiting backdoors for VCSP. We first present the formal definition of backdoors in the context of VCSP and describe how such backdoors once found, can be used to solve the VCSP instance. Subsequently, we show how to detect backdoors into a single tractable VCSP class with certain properties. In fact, our proof shows something stronger. That is, we show how to detect *heterogeneous* backdoors into a finite set of VCSP classes which satisfy these properties. The notion of heterogeneous backdoors is based on that introduced by Gaspers *et al.* [15]. For now, we proceed with the definition of a backdoor.

Definition 2. *Let $\mathcal{H}$ be a fixed class of VCSP instances over a domain D and let $\mathcal{P} = (V, D, \mathcal{C})$ be a VCSP instance. A* backdoor *into $\mathcal{H}$ is a subset $X \subseteq V$ such that for each assignment $\tau : X \to D$, the reduced instance $\mathcal{P}|_\tau$ is in $\mathcal{H}$.*

We note that this naturally corresponds to the notion of a *strong* backdoor in the context of Constraint Satisfaction and Satisfiability [27,28]; here we drop the adjective "strong" because the other kind of backdoors studied on these structures (so-called *weak* backdoors) do not seem to be useful in the general VCSP setting. Namely, in analogy to the CSP setting, one would define a weak backdoor of a VCSP instance $\mathcal{P} = (V, D, \mathcal{C})$ into $\mathcal{H}$ as a subset $X \subseteq V$ such that for some assignment $\tau : X \to D$ (i) the reduced instance $\mathcal{P}|_\tau$ is in $\mathcal{H}$ and (ii) τ can be extended to an assignment to V of minimum cost. However, in order to ensure (ii) we need to compare the cost of τ with the costs of all other assignments τ' to V. If X is not a strong backdoor, then some of the reduced instances $\mathcal{P}|_{\tau' \text{restricted to } X}$ will be outside of $\mathcal{H}$, and so in general we have no efficient way of determining a minimum cost assignment for it.

We begin by showing that small backdoors for globally tractable languages can always be used to efficiently solve VCSP instances as long the domain is finite (assuming such a backdoor is known).

Lemma 1. *Let $\mathcal{H}$ be a tractable class of* VCSP *instances over a finite domain D. There exists an algorithm which takes as input a* VCSP *instance $\mathcal{P}$ along with a backdoor X of $\mathcal{P} = (V, D, \mathcal{C})$ into $\mathcal{H}$, runs in time $\mathcal{O}^*(|D|^{|X|})$, and solves $\mathcal{P}$.*

Proof. Let $\mathcal{B}$ be a polynomial-time algorithm which solves every $\mathcal{P}$ in $\mathcal{H}$, i.e., outputs a minimum-cost assignment in $\mathcal{P}$; the existence of $\mathcal{B}$ follows by the tractability of $\mathcal{H}$. Consider the following algorithm $\mathcal{A}$. First, $\mathcal{A}$ branches on the at most $|D|^{|X|}$-many partial assignments of X. In each branch, $\mathcal{A}$ then applies the selected partial assignment τ to obtain the instance $\mathcal{P}|_\tau$ in linear time. In this branch, $\mathcal{A}$ proceeds by calling $\mathcal{B}$ on $\mathcal{P}|_\tau$, and stores the produced assignment along with its cost. After the branching is complete $\mathcal{A}$ reads the table of all of the at most $|D|^{|X|}$ assignments and costs outputted by $\mathcal{B}$, and selects one assignment (say α) with a minimum value (cost) a. Let τ be the particular partial assignment on X which resulted in the branch leading to α. $\mathcal{A}$ then outputs the assignment $\alpha \cup \tau$ along with the value (cost) a.   □

Already for crisp languages it is known that having a small backdoor does not necessarily allow for efficient (i.e., fixed-parameter) algorithms when the domain is not bounded. Specifically, the W[1]-hard k-clique problem can be encoded into a CSP with only k variables [23], which naturally contains a backdoor of size at most k for every crisp language under the natural assumption that the language contains the empty constraint. Hence the finiteness of the domain in Lemma 1 is a necessary condition for the statement to hold.

Next, we show that it is possible to find a small backdoor into VCSP[Γ] efficiently (or correctly determine that no such small backdoor exists) as long as Γ has two properties. First, Γ must be efficiently recognizable; it is easily seen that this condition is a necessary one, since detection of an empty backdoor is equivalent to determining whether the instance lies in VCSP[Γ]. Second, the arity of Γ must be bounded. This condition is also necessary since already in the more restricted CSP setting it was shown that backdoor detection for a wide range of natural crisp languages (of unbounded arity) is W[2]-hard [15].

Before we proceed, we introduce the notion of heterogeneous backdoors for VCSP which represent a generalization of backdoors into classes defined in terms of a single language. For languages $\Gamma_1, \ldots, \Gamma_\ell$, a heterogeneous backdoor is a backdoor into the class $\mathcal{H} = \text{VCSP}[\Gamma_1] \cup \cdots \cup \text{VCSP}[\Gamma_\ell]$; in other words, after each assignment to the backdoor variables, all cost functions in the resulting instance must belong to a language from our set. We now show that detecting small heterogenous backdoors is fixed-parameter tractable parameterized by the size of the backdoor.

Lemma 2. *Let $\Gamma_1, \ldots, \Gamma_\ell$ be efficiently recognizable languages over a domain D of size at most d and let q be a bound on the arity of Γ_i for every $i \in [\ell]$. There exists an algorithm which takes as input a VCSP instance $\mathcal{P}$ over D and an integer k, runs in time $O^*((\ell \cdot d \cdot (q+1))^k)$, and either outputs a backdoor X of $\mathcal{P}$ into $\text{VCSP}[\Gamma_1] \cup \cdots \cup \text{VCSP}[\Gamma_\ell]$ such that $|X| \leq k$ or correctly concludes that no such backdoor exists.*

Proof. The algorithm is a standard branching algorithm (see also [15]). Formally, the algorithm is called Detectbd, takes as input an instance $\mathcal{P} = (V, D, \mathcal{C})$, integer k, a set of variables B of size at most k and in time $O^*((\ell \cdot d \cdot (q+1))^k)$ either correctly concludes that $\mathcal{P}$ has no backdoor $Z \supseteq B$ of size at most k into $\text{VCSP}[\Gamma_1] \cup \cdots \cup \text{VCSP}[\Gamma_\ell]$ or returns a backdoor Z of $\mathcal{P}$ into $\text{VCSP}[\Gamma_1] \cup \cdots \cup \text{VCSP}[\Gamma_\ell]$ of size at most k. The algorithm is initialized with $B = \emptyset$.

In the base case, if $|B| = k$, and B is a backdoor of $\mathcal{P}$ into $\text{VCSP}[\Gamma_1] \cup \cdots \cup \text{VCSP}[\Gamma_\ell]$ then we return the set B. Otherwise, we return No. We now move to the description of the case when $|B| < k$.

In this case, if for every $\sigma : B \to D$ there is an $i \in [\ell]$ such that $\mathcal{P}|_\sigma \in \text{VCSP}[\Gamma_i]$, then it sets $Z = B$ and returns it. That is, if B is already found to be a backdoor of the required kind, then the algorithm returns B. Otherwise, it computes an assignment $\sigma : B \to D$ and valued constraints $c_1, \ldots, c_\ell$ in $\mathcal{P}|_\sigma$ such that for every $i \in [\ell]$, the cost function of c_i is not in Γ_i. Observe that for some σ, such a set of constraints must exist. Furthermore, since every Γ_i is

efficiently recognizable and B has size at most k, the selection of these valued constraints takes time $\mathcal{O}^*(d^k)$. The algorithm now constructs a set Y as follows. Initially, $Y = \emptyset$. For each $i \in [\ell]$, if the scope of the constraint c_i contains more than q variables then it adds to Y an arbitrary $q + 1$-sized subset of the scope of c_i. Otherwise, it adds to Y all the variables in the scope of c. This completes the definition of Y. Observe that any backdoor set for the given instance which contains B must also intersect Y. Hence the algorithm now branches on the set Y. Formally, for every $x \in Y$ it executes the recursive calls $\mathsf{Detectbd}(\mathcal{P}, k, B \cup \{x\})$. If for some $x \in Y$, the invoked call returned a set of variables, then it must be a backdoor set of the given instance and hence it is returned. Otherwise, the algorithm returns No.

Since the branching factor of this algorithm is at most $\ell \cdot (q + 1)$ and the set B, whose size is upper bounded by k, is enlarged with each recursive call, the number of nodes in the search tree is bounded by $\mathcal{O}((\ell \cdot (q+1))^k)$. Since the time spent at each node is bounded by $\mathcal{O}^*(d^k)$, the running time of the algorithm $\mathsf{Detectbd}$ is bounded by $\mathcal{O}^*((\ell \cdot (q + 1) \cdot d)^k)$. $\qquad\square$

Combining Lemmas 1 and 2, we obtain the main result of this section.

Corollary 1. *Let $\Gamma_1, \ldots, \Gamma_\ell$ be globally tractable and efficiently recognizable languages each of arity at most q over a domain of size d. There exists an algorithm which solves* VCSP *in time $O^*((\ell \cdot d \cdot (q + 1))^{k^2 + k})$, where k is the size of a minimum backdoor of the given instance into* $\mathrm{VCSP}[\Gamma_1] \cup \cdots \cup \mathrm{VCSP}[\Gamma_\ell]$.

4   Backdoors into Scattered Classes

Having established Corollary 1 and knowing that both the arity and domain restrictions of the language are necessary, it is natural to ask whether it is possible to push the frontiers of tractability for backdoors to more general classes of VCSP instances. In particular, there is no natural reason why the instances we obtain after each assignment into the backdoor should necessary always belong to the same language Γ even if Γ itself is one among several globally tractable languages. In fact, it is not difficult to show that as long as each "connected component" of the instance belongs to some tractable class after each assignment into the backdoor, then we can use the backdoor in a similar fashion as in Lemma 1. Such a generalization of backdoors from single languages to collections of languages has recently been obtained in the CSP setting [14] for conservative constraint languages. We proceed by formally defining these more general classes of VCSP instances, along with some other required notions.

4.1   Scattered Classes

A VCSP instance $(V, D, \mathcal{C})$ is *connected* if for each partition of its variable set into nonempty sets V_1 and V_2, there exists at least one constraint $c \in \mathcal{C}$ such that $\mathbf{var}(c) \cap V_1 \neq \emptyset$ and $\mathbf{var}(c) \cap V_2 \neq \emptyset$. A *connected component* of $(V, D, \mathcal{C})$ is a maximal connected subinstance $(V', D, \mathcal{C}')$ for $V' \subseteq V$, $\mathcal{C}' \subseteq \mathcal{C}$. These

notions naturally correspond to the connectedness and connected components of standard graph representations of VCSP instances.

Let $\Gamma_1, \ldots, \Gamma_d$ be languages. Then the *scattered class* $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_d)$ is the class of all instances $(V, D, \mathcal{C})$ which may be partitioned into pairwise variable disjoint subinstances $(V_1, D, \mathcal{C}_1), \ldots, (V_d, D, \mathcal{C}_d)$ such that $(V_i, D, \mathcal{C}_i) \in \mathrm{VCSP}[\Gamma_i]$ for each $i \in [d]$. Equivalently, an instance $\mathcal{P}$ is in $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_d)$ iff each connected component in $\mathcal{P}$ belongs to some $\mathrm{VCSP}[\Gamma_i]$, $i \in [d]$.

Lemma 3. *Let $\Gamma_1, \ldots, \Gamma_d$ be globally tractable languages. Then there exists a polynomial-time algorithm solving* VCSP *for all instances* $P \in \mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_d)$.

It is worth noting that while scattered classes on their own are a somewhat trivial extension of the tractable classes defined in terms of individual languages, backdoors into scattered classes can be much smaller than backdoors into each individual globally tractable language (or, more precisely, each individual class defined by a globally tractable language). That is because a backdoor can not only simplify cost functions to ensure they belong to a specific language, but it can also disconnect the instance into several "parts", each belonging to a different language, and furthermore the specific language each "part" belongs to can change for different assignments into the backdoor. As a simple example of this behavior, consider the boolean domain, let Γ_1 be the globally tractable crisp language corresponding to Horn constraints [24], and let Γ_2 be a globally tractable language containing only submodular cost functions [4]. It is not difficult to construct an instance $\mathcal{P} = (V_1 \cup V_2 \cup \{x\}, \{0, 1\}, \mathcal{C})$ such that (**a**) every assignment to x disconnects V_1 from V_2, (**b**) in $\mathcal{P}|_{x \mapsto 0}$, all valued constraints over V_1 are crisp Horn constraints and all valued constraints over V_2 are submodular, and (**c**) in $\mathcal{P}|_{x \mapsto 1}$, all valued constraints over V_1 are submodular and all valued constraints over V_2 are crisp Horn constraints. In the hypothetical example above, it is easy to verify that x is a backdoor into $\mathrm{VCSP}[\Gamma_1] \oplus \mathrm{VCSP}[\Gamma_2]$ but the instance does not have a small backdoor into neither $\mathrm{VCSP}[\Gamma_1]$ nor $\mathrm{VCSP}[\Gamma_2]$.

It is known that backdoors into scattered classes can be used to obtain fixed-parameter algorithms for CSP i.e., both finding and using such backdoors is FPT when dealing with crisp languages of bounded arity and domain size [14]. Crucially, these previous results relied on the fact that every crisp language of bounded arity and domain size is finite (which is not true for valued constraint languages in general). We formalize this below.

Theorem 1 ([14, Lemma 1.1]). *Let $\Gamma_1, \ldots, \Gamma_\ell$ be globally tractable conservative crisp languages over a domain D, with each language having arity at most q and containing at most p relations. There exists a function f and an algorithm solving* VCSP *in time $\mathcal{O}^*(f(\ell, |D|, q, k, p))$, where k is the size of a minimum backdoor into $\mathrm{VCSP}[\Gamma_1] \oplus \cdots \oplus \mathrm{VCSP}[\Gamma_\ell]$.*

Observe that in the above theorem, when q and $|D|$ are bounded, p is immediately bounded. However, it is important that we formulate the running time

of the algorithm in this form because in the course of our application, these parameters have to be bounded separately. Our goal for the remainder of this section is to extend Theorem 1 in the VCSP setting to also cover infinite globally tractable languages (of bounded arity and domain size). Before proceeding, it will be useful to observe that if each $\Gamma_1, \ldots, \Gamma_\ell$ is globally tractable, then the class $\mathrm{VCSP}[\Gamma_1] \oplus \cdots \oplus \mathrm{VCSP}[\Gamma_\ell]$ is also tractable (since each connected component can be resolved independently of the others).

4.2 Finding Backdoors to Scattered Classes

In this subsection, we prove that finding backdoors for VCSP into scattered classes is fixed-parameter tractable. This will then allow us to give a proof of our main theorem, stated below.

Theorem 2. *Let $\Delta_1, \ldots, \Delta_\ell$ be conservative, globally tractable and efficiently recognizable languages over a finite domain and having constant arity. Then VCSP is fixed-parameter tractable parameterized by the size of a smallest backdoor of the given instance into $\mathrm{VCSP}(\Delta_1) \oplus \cdots \oplus \mathrm{VCSP}(\Delta_\ell)$.*

Recall that the closure of a conservative and globally tractable language under partial assignments is also a globally tractable language. Furthermore, every backdoor of the given instance into $\mathrm{VCSP}(\Delta_1) \oplus \cdots \oplus \mathrm{VCSP}(\Delta_\ell)$ is also a backdoor into $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$ where Γ_i is the closure of Δ_i under partial assignments. Due to Lemma 1, it follows that it is sufficient to compute a backdoor of small size into the scattered class $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$ where each Γ_i is closed under partial assignments.

Our strategy for finding backdoors to scattered classes defined in terms of (potentially infinite) globally tractable languages relies on a two-phase transformation of the input instance. In the first phase (Lemma 4), we show that for every choice of $\Gamma_1, \ldots, \Gamma_d$ (each having bounded domain size and arity), we can construct a set of finite languages $\Gamma'_1, \ldots, \Gamma'_d$ and a new instance $\mathcal{P}'$ such that there is a one-to-one correspondence between backdoors of $\mathcal{P}$ into $\Gamma_1 \oplus \cdots \oplus \Gamma_d$ and backdoors of $\mathcal{P}'$ into $\Gamma'_1 \oplus \cdots \oplus \Gamma'_d$. This allows us to restrict ourselves to only the case of finite (but not necessarily crisp) languages as far as backdoor detection is concerned. In the second phase (Lemma 5), we transform the instance and languages one more time to obtain another instance $\mathcal{P}''$ along with finite crisp languages $\Gamma''_1, \ldots, \Gamma''_d$ such that there is a one-to-one correspondence between the backdoors of $\mathcal{P}''$ and backdoors of $\mathcal{P}'$. We crucially note that the newly constructed instances are equivalent *only* with respect to backdoor detection; there is no correspondence between the solutions of these instances.

Before proceeding, we introduce a natural notion of replacement of valued constraints which is used in our proofs.

Definition 3. *Let $\mathcal{P} = (V, D, \mathcal{C})$ be a VCSP instance and let $c = (\boldsymbol{x}, \varphi) \in \mathcal{C}$. Let φ' be a cost function over D with the same arity as φ. Then the operation of replacing φ in c with φ' results in a new instance $\mathcal{P}' = (V, D, (\mathcal{C} \setminus \{c\}) \cup \{(\boldsymbol{x}, \varphi')\})$.*

Lemma 4. *Let $\Gamma_1, \ldots, \Gamma_\ell$ be efficiently recognizable languages closed under partial assignments, each of arity at most q over a domain D of size d. There exists an algorithm which takes as input a VCSP instance $\mathcal{P} = (V, D, \mathcal{C})$ and an integer k, runs in time $\mathcal{O}^*(f(\ell, d, k, q))$ for some function f and either correctly concludes that $\mathcal{P}$ has no backdoor into $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$ of size at most k or outputs a VCSP instance $\mathcal{P}' = (V, D', \mathcal{C}')$ and languages $\Gamma_1', \ldots, \Gamma_\ell'$ with the following properties.*

1. *For each $i \in [\ell]$, the arity of Γ_i' is at most q*
2. *For each $i \in [\ell]$, Γ_i' is over D' and $D' \subseteq D$*
3. *Each of the languages $\Gamma_1', \ldots, \Gamma_\ell'$ is closed under partial assignments and contains at most $g(\ell, d, k, q)$ cost functions for some function g.*
4. *For each $X \subseteq V$, X is a minimal backdoor of $\mathcal{P}$ into $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$ of size at most k if and only if X is a minimal backdoor of $\mathcal{P}'$ into $\mathrm{VCSP}(\Gamma_1') \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell')$ of size at most k.*

Proof. We will first define a function mapping the valued constraints in $\mathcal{C}$ to a finite set whose size depends only on ℓ, d, k and q. Subsequently, we will show that every pair of constraints in $\mathcal{C}$ which are mapped to the same element of this set are, for our purposes (locating a backdoor), interchangeable. We will then use this observation to define the new instance $\mathcal{P}'$ and the languages $\Gamma_1', \ldots, \Gamma_{\ell'}'$. To begin with, observe that if the arity of a valued constraint in $\mathcal{P}$ is at least $q+k+1$, then $\mathcal{P}$ has no backdoor of size at most k into $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$. Hence, we may assume without loss of generality that the arity of every valued constraint in $\mathcal{P}$ is at most $q + k$.

Let $\mathcal{F}$ be the set of all functions from $[q + k] \times 2^{[q+k]} \times D^{[q+k]} \to 2^{[\ell]} \cup \{\bot\}$, where $\bot$ is used a special symbol expressing that $\mathcal{F}$ is "out of bounds." Observe that $|\mathcal{F}| \leq \eta(\ell, d, k, q) = (2^\ell + 1)^{(2d)^{(q+k)+\log(q+k)}}$. We will now define a function $\mathsf{Type} : \mathcal{C} \to \mathcal{F}$ as follows. We assume without loss of generality that the variables in the scope of each constraint in $\mathcal{C}$ are numbered from 1 to $|\mathbf{var}(c)|$ based on their occurrence in the tuple $\boldsymbol{x}$ where $c = (\boldsymbol{x}, \varphi)$. Furthermore, recall that $|\mathbf{var}(c)| \leq q + k$. For $c \in \mathcal{C}$, we define $\mathsf{Type}(c) = \delta \in \mathcal{F}$ where δ is defined as follows. Let $r \leq q + k$, $Q \subseteq [q + k]$ and $\gamma : [q + k] \to D$. Let $\gamma[Q \cap [r]]$ denote the restriction of γ to the set $Q \cap [r]$. Furthermore, recall that $c|_{\gamma[Q \cap [r]]}$ denotes the valued constraint resulting from applying the partial assignment γ on the variables of c corresponding to all those indices in $Q \cap [r]$.

Then, $\delta(r, Q, \gamma) = \bot$ if $r \neq |\mathbf{var}(c)|$. Otherwise, $\delta(r, Q, \gamma) = L \subseteq [\ell]$ where $i \in [\ell]$ is in L if and only if $c|_{\gamma[Q \cap [r]]} \in \mathrm{VCSP}(\Gamma_i)$. This completes the description of the function Type; observe that $\mathsf{Type}(c)$ can be computed in time which is upper-bounded by a function of ℓ, d, k, q.

For every $\delta \in \mathcal{F}$, if there is a valued constraint $c \in \mathcal{C}$ such that $\mathsf{Type}(c) = \delta$, we pick and fix one arbitrary such valued constraint $c_\delta^* = (\boldsymbol{x_\delta^*}, \varphi_\delta^*)$. We now proceed to the definition of the instance $\mathcal{P}'$ and the languages $\Gamma_1', \ldots, \Gamma_{\ell'}'$.

Observe that for 2 constraints $c = (\boldsymbol{x_1}, \varphi), c' = (\boldsymbol{x'}, \varphi') \in \mathcal{C}$, if $\mathsf{Type}(c) = \mathsf{Type}(c')$ then $|\mathbf{var}(c)| = |\mathbf{var}(c')|$. Hence, the notion of *replacing* φ in c with φ' is well-defined (see Definition 3). We define the instance $\mathcal{P}'$ as the instance

obtained from $\mathcal{P}$ by replacing each $c = (\boldsymbol{x}, \varphi) \in \mathcal{C}$ with the constraint $(\boldsymbol{x}, \varphi_\delta^*)$ where $\delta = \mathsf{Type}(c)$.

For each $i \in [\ell]$ and cost function $\varphi \in \Gamma_i$, we add φ to the language Γ_i' if and only if for some $\delta \in \mathcal{F}$ and some set $Q \subseteq \mathbf{var}(c_\delta^*)$ and assignment $\gamma : Q \to D$, the constraint $c|_{\gamma[Q]} = (\boldsymbol{x} \setminus Q, \varphi)$. Clearly, for every $i \in [\ell]$, $|\Gamma_i'| \leq d^q \cdot |\mathcal{F}| \leq d^q \cdot \eta(\ell, d, k, q)$. Finally, for each Γ_i', we compute the closure of Γ_i' under partial assignments and add each relation from this closure into Γ_i'. Since the size of each Γ_i' is bounded initially in terms of ℓ, d, k, q, computing this closure can be done in time $\mathcal{O}^*(\lambda(\ell, d, k, q))$ for some function λ. Since each cost function has arity q and domain D, the size of the final language Γ_i' obtained after this operation is blown up by a factor of at most d^q, implying that in the end, $|\Gamma_i'| \leq d^{2q} \cdot |\mathcal{F}| \leq d^{2q} \cdot \eta(\ell, d, k, q)$.

Now, observe that the first two statements of the lemma follow from the definition of the languages $\{\Gamma_i'\}_{i \in [\ell]}$. Furthermore, the number of cost functions in each Γ_i' is bounded by $d^q \cdot \eta(\ell, d, k, q)$, and so the third statement holds as well. Therefore, it only remains to prove the final statement of the lemma. Before we do so, we state a straightforward consequence of the definition of $\mathcal{P}'$.

Observation 1. *For every* $Y \subseteq V$, $\gamma : Y \to D$ *and connected component* $\mathcal{H}'$ *of* $\mathcal{P}'|_\gamma$, *there is a connected component* $\mathcal{H}$ *of* $\mathcal{P}|_\gamma$ *and a bijection* $\psi : \mathcal{H} \to \mathcal{H}'$ *such that for every* $c \in \mathcal{H}$, $\mathsf{Type}(c) = \mathsf{Type}(\psi(c))$. *Furthermore, for every* $c = (\boldsymbol{x}, \varphi) \in \mathcal{H}$, *the constraint* $\psi(c)$ *is obtained by replacing* φ *in* c *with* $\varphi_{\mathsf{Type}(c)}^*$.

We now return to the proof of Lemma 4. Consider the forward direction and let X be a backdoor of size at most k for $\mathcal{P}$ into $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$ and suppose that X is *not* a backdoor for $\mathcal{P}'$ into $\mathrm{VCSP}(\Gamma_1') \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell')$. Then, there is an assignment $\gamma : X \to D$ such that for some connected component $\mathcal{H}'$ of $\mathcal{P}'|_\gamma$, there is no $i \in \ell$ such that all constraints in $\mathcal{H}'$ lie in $\mathrm{VCSP}(\Gamma_i')$. By Observation 1 above, there is a connected component $\mathcal{H}$ in $\mathcal{P}|_\gamma$ and a bijection $\psi : \mathcal{H} \to \mathcal{H}'$ such that for every $c \in \mathcal{H}$, $\mathsf{Type}(c) = \mathsf{Type}(\psi(c))$. Since X is a backdoor for $\mathcal{P}$, there is a $j \in \ell$ such that all constraints in $\mathcal{H}$ lie in $\mathrm{VCSP}(\Gamma_j)$. Pick an arbitrary constraint $c = (\boldsymbol{x}, \varphi) \in \mathcal{H}$. Let $c' = (\boldsymbol{x}, \varphi_{\mathsf{Type}(c)}^*)$ be the constraint $\psi(c)$. By definition of $\varphi_{\mathsf{Type}(c)}^*$ it follows that $c'|_\gamma \in \mathrm{VCSP}(\Gamma_j)$. The fact that this holds for an arbitrary constraint in $\mathcal{H}$ along with the fact that ψ is a bijection implies that every constraint in $\mathcal{H}'$ is in fact in $\mathrm{VCSP}(\Gamma_j')$, a contradiction. The argument in the converse direction is symmetric. This completes the proof of the final statement of the lemma.

The time taken to compute $\mathcal{P}'$ and the languages $\Gamma_1', \ldots, \Gamma_\ell'$ is dominated by the time required to compute the function Type. Since the languages $\Gamma_1, \ldots, \Gamma_\ell$ are efficiently recognizable, this time is bounded by $\mathcal{O}^*(|\mathcal{F}|)$, completing the proof of the lemma. $\square$

Lemma 5. *Let* $\Gamma_1, \ldots, \Gamma_\ell$ *be efficiently recognizable languages closed under partial assignments, each of arity at most* q *over a domain* D *of size* d. *Let* $\mathcal{P}' = (V, D', \mathcal{C}')$ *be the* VCSP *instance and let* $\Gamma_1', \ldots, \Gamma_\ell'$ *be languages returned by the algorithm of Lemma 4 on input* $\mathcal{P}$ *and* k. *There exists an algorithm which takes as input* $\mathcal{P}'$, *these languages and* k, *runs in time* $\mathcal{O}^*(f(\ell, d, k, q))$ *for some*

function f and outputs a CSP *instance* $\mathcal{P}'' = (V'' \supseteq V, D'', \mathcal{C}'')$ *and crisp languages* $\Gamma_1'', \dots, \Gamma_\ell''$ *with the following properties.*

1. *For each $i \in [\ell]$, the arity of Γ_i'' is at most $q + 1$*
2. *$D'' \supseteq D$ and $|D''| \le \beta(q, d, k)$ for some function β.*
3. *The number of relations in each of the languages $\Gamma_1'', \dots, \Gamma_\ell''$ is at most $\alpha(q, d, k)$ for some function α.*
4. *if X is a minimal backdoor of arity at most k of $\mathcal{P}''$ into $\mathrm{CSP}(\Gamma_1'') \oplus \cdots \oplus \mathrm{CSP}(\Gamma_\ell'')$, then $X \subseteq V$.*
5. *For each $X \subseteq V$, X is a minimal backdoor of $\mathcal{P}'$ into $\mathrm{VCSP}(\Gamma_1') \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell')$ if and only if X is a minimal backdoor of $\mathcal{P}''$ into $\mathrm{CSP}(\Gamma_1'') \oplus \cdots \oplus \mathrm{CSP}(\Gamma_\ell'')$.*

Proof. We propose a fixed-parameter algorithm $\mathcal{A}$, and show that it has the claimed properties. It will be useful to recall that we do not distinguish between crisp cost functions and relations. We also formally assume that D' does not intersect the set of rationals $\mathcal{Q}$; if this is not the case, then we simply rename elements of D' to make sure that this holds. Within the proof, we will use $\boldsymbol{a} \circ b$ to denote the concatenation of vector $\boldsymbol{a}$ by element b.

First, let T_i be the set of all values which are returned by at least one cost function from Γ_i', $i \in [\ell]$, for at least one input. Let $T = \bigcup_{i \in [\ell]} T_i$. Observe that $|T|$ is upper-bounded by the size, domain and arity of our languages. Let us now set $D'' = D' \cup T \cup \epsilon$. Intuitively, our goal will be to represent the cost function in each valued constraint in $\mathcal{P}'$ by a crisp cost function with one additional variable which ranges over T, where T corresponds to a specific value which occurs in one of our base languages. Note that this satisfies Condition 2 of the lemma, and that T can be computed in linear time from the cost tables of $\Gamma_1', \dots, \Gamma_\ell'$. We will later construct $k + 1$ such representations (each with its own additional variable) to ensure that the additional variables are never selected by minimal backdoors.

Next, for each language Γ_i', $i \in [\ell]$, we compute a new crisp language Γ_i'' as follows. For each $\varphi \in \Gamma_i'$ of arity t, we add a new relation ψ of arity $t + 1$ into Γ_i'', and for each tuple $(x_1, \dots, x_t)$ of elements from D' we add the tuple $(x_1, \dots, x_t, \varphi(x_1, \dots, x_t))$ into ψ; observe that this relation exactly corresponds to the cost table of φ. We then compute the closure of Γ_i'' under partial assignments and add each relation from this closure into Γ_i''. Observe that the number of relations in Γ_i'' is bounded by a function of $|T|$ and $|\Gamma_i'|$, and furthermore the number of tuples in each relation is upper-bounded by $q^{|D'|}$, and so Conditions 1 and 3 of the lemma hold. The construction of each Γ_i'' from Γ_i' can also be done in linear time from the cost tables of $\Gamma_1', \dots, \Gamma_\ell'$.

Finally, we construct a new instance $\mathcal{P}'' = (V'', D'', \mathcal{C}'')$ from $\mathcal{P}' = (V, D', \mathcal{C}')$ as follows. At the beginning, we set $V'' := V$. For each $c' = (\boldsymbol{x}', \varphi') \in \mathcal{C}'$, we add $k + 1$ unique new variables $v_{c'}^1, \dots, v_{c'}^{k+1}$ into V'' and add $k + 1$ constraints $c''^1, \dots, c''^{k+1}$ into $\mathcal{C}''$. For $i \in [k + 1]$, each $c''^i = (\boldsymbol{x}' \circ v_{c'}^i, \psi'')$ where ψ'' is a relation that is constructed similarly as the relations in our new languages Γ_i'' above. Specifically, for each tuple $(x_1, \dots, x_t)$ of elements from D' we add

the tuple $(x_1, \ldots, x_t, \varphi'(x_1, \ldots, x_t))$ into ψ'', modulo the following exception. If $\varphi'(x_1, \ldots, x_t) \notin D''$, then we instead add the tuple $(x_1, \ldots, x_t, \epsilon)$ into ψ''. Clearly, the construction of our new instance $\mathcal{P}''$ takes time at most $\mathcal{O}(|\mathcal{C}'| + q^{|D'|})$. This concludes the description of $\mathcal{A}$.

It remains to argue that Conditions 4 and 5 of the lemma hold. First, consider a minimal backdoor X of size at most k of $\mathcal{P}''$ into $\mathrm{CSP}[\Gamma_1''] \oplus \cdots \oplus \mathrm{CSP}[\Gamma_\ell'']$, and assume for a contradiction that there exists some $c' = (\boldsymbol{x}', \varphi') \in \mathcal{C}'$ and $i \in [k+1]$ such that $v_{c'}^i \in X$. First, observe that this cannot happen if the whole scope of c''^i lies in X. By the size bound on X, there exists $j \in [k+1]$ such that $v_{c'}^j \notin X$. Then for each partial assignment τ of X, the relation φ'' in c''^j belongs to the same globally tractable language as the rest of the connected component of $\mathcal{P}''$ containing the scope of c'' (after applying τ). Since the relation φ'' in c''^j is precisely the same as in c''^i and the scope of c''^i must lie in the same connected component as that of c''^j, it follows that $X \setminus \{v_{x'}^i\}$ is also a backdoor of $\mathcal{P}''$ into $\mathrm{CSP}(\Gamma_1'') \oplus \cdots \oplus \mathrm{CSP}(\Gamma_\ell'')$. However, this contradicts the minimality of X.

Finally, for Condition 5, consider an arbitrary backdoor X of $\mathcal{P}'$ into $\mathrm{VCSP}(\Gamma_1') \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell')$, and let us consider an arbitrary assignment from X to D''. It will be useful to note that while the contents of relations and/or cost functions in individual (valued) constraints depend on the particular choice of the assignment to X, which variables actually occur in individual components depends only on the choice of X and remains the same for arbitrary assignments.

Now observe that each connected component $\mathcal{P}^{\mathrm{CSP}}$ of $\mathcal{P}''$ after the application of the (arbitrarily chosen) assignment will fall into one of the following two cases. $\mathcal{P}^{\mathrm{CSP}}$ could contain a single variable $v_{c'}$ with a single constraint whose relation lies in every language Γ_i'', $i \in [\ell]$; this occurs precisely when the whole scope of a valued constraint $c' \in \mathcal{C}'$ lies in X, and the relation will either contain a singleton element from T or be the empty relation. In this case, we immediately conclude that $\mathcal{P}^{\mathrm{CSP}} \in \mathrm{CSP}(\Gamma_i)$ for each $i \in [\ell]$.

Alternatively, $\mathcal{P}^{\mathrm{CSP}}$ contains at least one variable $v \in V$. Let $\mathcal{P}^{\mathrm{VCSP}}$ be the unique connected component of $\mathcal{P}'$ obtained after the application of an arbitrary assignment from X to D' which contains v. Observe that the variable sets of $\mathcal{P}^{\mathrm{CSP}}$ and $\mathcal{P}^{\mathrm{VCSP}}$ only differ in the fact that $\mathcal{P}^{\mathrm{CSP}}$ may contain some of the newly added variables $v_{c'}$ for various constraints c'. Now let us consider a concrete assignment τ from X to D' along with an $i \in [\ell]$ such that after the application of τ, the resulting connected component $\mathcal{P}^{\mathrm{VCSP}}$ belongs to $\mathrm{VCSP}(\Gamma_i')$. It follows by our construction that applying the same assignment τ in $\mathcal{P}''$ will result in a connected component $\mathcal{P}^{\mathrm{CSP}}$ corresponding to $\mathcal{P}^{\mathrm{VCSP}}$ such that $\mathcal{P}^{\mathrm{VCSP}} \in \mathrm{CSP}(\Gamma_i'')$; indeed, whenever Γ_i' contains an arbitrary cost function $\varphi(\boldsymbol{x}) = \beta$, the language Γ_i'' will contain the relation $(\boldsymbol{x} \circ \beta)$.

By the above, the application of an assignment from X to D' in $\mathcal{P}''$ will indeed result in an instance in $\mathrm{CSP}(\Gamma_1'' \oplus \cdots \oplus \Gamma_\ell'')$. But recall that the domain of $\mathcal{P}''$ is D'', which is a superset of D'; we need to argue that the above also holds for assignments τ from X to D''. To this end, consider an arbitrary such τ and let τ_0 be an arbitrary assignment from X to D' which matches τ on all mappings into D'. Let us compare the instances $\mathcal{P}''_{\tau_0}$ and $\mathcal{P}''_{\tau}$. By our construction

of $\mathcal{P}'$, whenever τ maps at least one variable from the scope of some constraint c'' to $D'' \setminus D'$, the resulting relation will be the empty relation. It follows that each constraint in $\mathcal{P}''_\tau$ will either be the same as in $\mathcal{P}''_\tau$, or will contain the empty relation. But since the empty relation is included in every language $\Gamma''_1, \ldots, \Gamma''_\ell$, we conclude that each connected component of $\mathcal{P}''_\tau$ must belong to at least one language Γ''_i, $i \in [\ell]$. This shows that X must also be a backdoor of $\mathcal{P}''$ into $\mathrm{CSP}[\Gamma''_1] \oplus \cdots \oplus \mathrm{CSP}[\Gamma''_\ell]$.

For the converse direction, consider a minimal backdoor X of $\mathcal{P}''$ into $\mathrm{CSP}[\Gamma''_1] \oplus \cdots \oplus \mathrm{CSP}[\Gamma''_\ell]$. Since we already know that Condition 4 holds, X must be a subset of V. The argument from the previous case can then simply be reversed to see that X will also be a backdoor of $\mathcal{P}'$ into $\mathrm{VCSP}[\Gamma'_1] \oplus \cdots \oplus \mathrm{VCSP}[\Gamma'_\ell]$; in fact, the situation in this case is much easier since only assignments into D' need to be considered.

Summarizing, we gave a fixed-parameter algorithm and then showed that it satisfies each of the required conditions, and so the proof is complete.   □

We are now ready to prove Theorem 2, which we restate for the sake of convenience.

Theorem 2. *Let $\Delta_1, \ldots, \Delta_\ell$ be conservative, globally tractable and efficiently recognizable languages over a finite domain and having constant arity. Then VCSP is fixed-parameter tractable parameterized by the size of a smallest backdoor of the given instance into $\mathrm{VCSP}(\Delta_1) \oplus \cdots \oplus \mathrm{VCSP}(\Delta_\ell)$.*

Proof. For each $i \in [\ell]$, let Γ_i denote the closure of Δ_i under partial assignments. Observe that every backdoor of the given instance into $\mathrm{VCSP}(\Delta_1) \oplus \cdots \oplus \mathrm{VCSP}(\Delta_\ell)$ is also a backdoor into $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$. Furthermore, each $\mathrm{VCSP}(\Gamma_i)$ is tractable since $\mathrm{VCSP}(\Delta_1)$ is tractable and conservative. Hence, it is sufficient to compute and use a backdoor of size at most k into $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$.

The claimed algorithm has two parts. The first one is finding a backdoor into $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$ and the second one is using the computed backdoor to solve VCSP. Given an instance $\mathcal{P}$ and k, we first execute the algorithm of Lemma 4 to compute the instance $\mathcal{P}'$, and the languages $\Gamma'_1, \ldots, \Gamma'_\ell$ with the properties stated in the lemma. We then execute the algorithm of Lemma 5 with input $\mathcal{P}'$, k, and $\Gamma'_1, \ldots, \Gamma'_\ell$ to compute the CSP instance $\mathcal{P}''$ and crisp languages $\Gamma''_1, \ldots, \Gamma''_\ell$ with the stated properties. Following this, we execute the algorithm of Theorem 1 with input $\mathcal{P}'', k$. If this algorithm returns No then we return No as well. Otherwise we return the set returned by this algorithm as a backdoor of size at most k for the given instance $\mathcal{P}$. Finally, we use the algorithm of Lemma 1 with $\mathcal{H}$ set to be the class $\mathrm{VCSP}(\Gamma_1) \oplus \cdots \oplus \mathrm{VCSP}(\Gamma_\ell)$, to solve the given instance.

The correctness as well as running time bounds follow from those of Lemmas 4 and 5, Theorem 1, and Lemma 1. This completes the proof of the theorem.

5   Concluding Remarks

We have introduced the notion of backdoors to the VCSP setting as a means for augmenting a class of globally tractable VCSP instances to instances that are outside the class but of small distance to the class. We have presented fixed-parameter tractability results for solving VCSP instances parameterized by the size of a smallest backdoor into a (possibly scattered and heterogeneous) tractable class satisfying certain natural properties.

Our work opens up several avenues for future research. Since our main objective was to establish the fixed-parameter tractability of this problem, we have not attempted to optimize the runtime bounds for finding backdoors to scattered classes. As a result, it is quite likely that a more focussed study of scattered classes arising from specific constraint languages will yield a significantly better runtime. A second interesting direction would be studying the parameterized complexity of detection of backdoors into tractable VCSP classes that are characterized by specific fractional polymorphisms.

References

1. Bessiere, C., Carbonnel, C., Hebrard, E., Katsirelos, G., Walsh, T.: Detecting and exploiting subproblem tractability. In: Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, 3–9 August. IJCAI/AAAI (2013)
2. Carbonnel, C., Cooper, M.C.: Tractability in constraint satisfaction problems: a survey. Constraints **21**(2), 115–144 (2016)
3. Carbonnel, C., Cooper, M.C., Hebrard, E.: On backdoors to tractable constraint languages. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 224–239. Springer, Heidelberg (2014)
4. Cohen, D.A., Jeavons, P.G., Zivny, S.: The expressive power of valued constraints: hierarchies and collapses. Theoret. Comput. Sci. **409**(1), 137–153 (2008)
5. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, New York (2015)
6. Dilkina, B.N., Gomes, C.P., Sabharwal, A.: Tradeoffs in the complexity of backdoor detection. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 256–270. Springer, Heidelberg (2007)
7. Dilkina, B.N., Gomes, C.P., Sabharwal, A.: Backdoors in the context of learning. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 73–79. Springer, Heidelberg (2009)
8. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer, New York (2013)
9. Dvorák, W., Ordyniak, S., Szeider, S.: Augmenting tractable fragments of abstract argumentation. Artif. Intell. **186**, 157–173 (2012)
10. Fichte, J.K., Szeider, S.: Backdoors to normality for disjunctive logic programs. ACM Trans. Comput. Log. **17**(1), 1–23 (2015)
11. Fichte, J.K., Szeider, S.: Backdoors to tractable answer set programming. Artif. Intell. **220**, 64–103 (2015)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series, vol. XIV. Springer, Berlin (2006)

13. Ganian, R., Ordyniak, S.: The complexity landscape of decompositional parameters for ILP. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI Press (to appear, 2016)

14. Ganian, R., Ramanujan, M.S., Szeider, S.: Discovering archipelagos of tractability for constraint satisfaction and counting. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, 10–12 January, pp. 1670–1681 (2016)

15. Gaspers, S., Misra, N., Ordyniak, S., Szeider, S., Živný, S.: Backdoors into heterogeneous classes of SAT and CSP. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, Québec City, Québec, Canada, 27–31 July, pp. 2652–2658. AAAI Press (2014)

16. Gaspers, S., Szeider, S.: Backdoors to satisfaction. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond. LNCS, vol. 7370, pp. 287–317. Springer, Heidelberg (2012)

17. Jeavons, P., Krokhin, A.A., Živný, S.: The complexity of valued constraint satisfaction. Bull. Eur. Assoc. Theoret. Comput. Sci. **113** (2014)

18. Kolmogorov, V., Živný, S.: The complexity of conservative valued CSPs. J. ACM **60**(2), Art. 10, 38 (2013)

19. Krokhin, A., Bulatov, A., Jeavons, P.: The complexity of constraint satisfaction: an algebraic approach. In: Structural Theory of Automata, Semigroups and Universal Algebra, Montreal, 2003. NATO Science Series II: Mathematics, Physics, and Chemistry, vol. 207, pp. 181–213 (2005)

20. LeBras, R., Bernstein, R., Gomes, C.P., Selman, B., van Dover, R.B.: Crowdsourcing backdoor identification for combinatorial optimization. In Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, 3–9 August 2013

21. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford (2006)

22. Nishimura, N., Ragde, P., Szeider, S.: Detecting backdoor sets with respect to Horn and binary clauses. In: Proceedings of Seventh International Conference on Theory and Applications of Satisfiability Testing, SAT 2004, Vancouver, BC, Canada, 10–13 May, pp. 96–103 (2004)

23. Papadimitriou, C.H., Yannakakis, M.: On the complexity of database queries. J. Comput. Syst. Sci. **58**(3), 407–427 (1999)

24. Schaefer, T.J.: The complexity of satisfiability problems. In: Conference Record of the Tenth Annual ACM Symposium on Theory of Computing, San Diego, Calif., 1978, pp. 216–226. ACM (1978)

25. Thapper, J., Živný, S.: Necessary conditions for tractability of valued CSPs. SIAM J. Discrete Math. **29**(4), 2361–2384 (2015)

26. Živný, S.: The Complexity of Valued Constraint Satisfaction Problems. Cognitive Technologies. Springer, New York (2012)

27. Williams, R., Gomes, C., Selman, B.: Backdoors to typical case complexity. In: Gottlob, G., Walsh, T. (eds.) Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003, pp. 1173–1178. Morgan Kaufmann (2003)

28. Williams, R., Gomes, C., Selman, B.: On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In: Informal Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, SAT 2003 S. Margherita Ligure - Portofino, Italy, 5–8 May, pp. 222–230 (2003)

Monte-Carlo Tree Search for the Maximum Satisfiability Problem

Jack Goffinet and Raghuram Ramanujan[(✉)]

Department of Mathematics and Computer Science, Davidson College,
Davidson, NC 28035, USA
{jagoffinet,raramanujan}@davidson.edu

Abstract. Incomplete algorithms for the Maximum Satisfiability (MaxSAT) problem use a hill climbing approach in tandem with various mechanisms that prevent search stagnation. These solvers' conflicting goals of maintaining search mobility while discovering high quality solutions constitute an exploration-exploitation dilemma, a problem which has been tackled with great success in recent years using Monte-Carlo Tree Search (MCTS) methods. We apply MCTS to the domain of MaxSAT using various stochastic local search (SLS) algorithms for leaf node value estimation, thus offering a novel hybrid alternative to established complete and incomplete solution techniques. Our algorithm outperforms baseline SLS algorithms like Walksat and Novelty on most problem instances from the 2015 MaxSAT Evaluation. It also outdoes CCLS, a state-of-the-art incomplete MaxSAT solver, on a number of challenging industrial instances from the 2015 MaxSAT Evaluation.

1 Introduction

A canonical NP-complete problem, Boolean Satisfiability (SAT), has attracted a tremendous amount of attention from researchers in the A.I. and broader computer science community over the years. The Maximum Satisfiability problem (MaxSAT) is a generalization of SAT: in this setting, the goal is to find a truth assignment that satisfies the maximum number of clauses (i.e., constraints) in a given formula. Modern MaxSAT solvers generally fall into one of two categories. *Complete* solvers systematically explore the problem search space and can find provably optimal solutions to a given instance. Typically, they are implemented using branch-and-bound techniques, or make iterative calls to a SAT solver [19]. In contrast, *incomplete* solvers start with a random truth assignment and attempt to discover better assignments by interleaving greedy and random walk steps in the search space according to some policy. The WalkSAT algorithm [23] is the exemplar of this class of stochastic local search (SLS) methods. Their memorylessness means that these algorithms are incapable of proving that a solution is optimal. Incomplete SLS solvers are nonetheless popular as they are faster than their complete counterparts. Moreover, SLS algorithms are able to find approximate solutions to large problem instances that are beyond the capabilities of complete MaxSAT solvers.

SLS algorithms work by using various properties of the underlying formula as a guiding gradient. For example, WalkSAT uses the number of unsatisfied clauses, while modern solvers like CCLS [16] use more intricate strategies such as configuration checking [4]. However, this greedy approach, which is beneficial for refining the quality of a solution, also often leads the solver into "basins" in the search space that contain suboptimal solutions and from which escape is difficult [10]. Authoring an effective SLS algorithm thus requires careful management of this tension between preventing search stagnation and ensuring the reachability of high-quality solutions. We observe that this can be viewed as an example of the exploration-exploitation dilemma that is well-studied in the reinforcement learning community [27]. In recent years, Monte-Carlo Tree Search (MCTS) methods have proven to be an effective option for addressing this dilemma, particularly within the setting of sequential decision-making problems.

MCTS methods, such as the Upper Confidence bounds applied to Trees (UCT) algorithm [13], first came to the attention of the wider A.I. community due to their startling success in some challenging adversarial planning domains, most notably Go [8]. In the intervening years, they have been successfully used to advance the state-of-the-art in several other domains, including other games [2,3,5], probabilistic single-agent planning [12], and planning in partially observable settings [26]. In all these domains, a balance needs to be struck between verifying the value of known actions versus evaluating the potential of under-explored actions, when under computational constraints. Algorithms like UCT approach this tradeoff in a theoretically sound way, by modeling the search process as a sequence of plays of a system of heirarchically organized multi-armed bandits [13]. In this paper, we investigate the potential of UCT in the domain of MaxSAT. In particular, we present a novel hybrid algorithm named UCTMAXSAT that combines aspects of both complete and incomplete solvers. Our algorithm maintains a search tree that is methodically explored. An SLS algorithm is used to estimate the value of the leaf nodes in this tree and guides its future expansion. We find that our algorithm outperforms the underlying SLS algorithms on a wide variety of benchmark instances.

2   Related Work

Several attempts have been made in the past to combine aspects of the complete and incomplete paradigms in an attempt to produce strong SAT solvers. In 1998, Mazure et al. studied the use of an SLS algorithm to inform the variable selection strategy in a complete solver [17], while in 2002, Habet et al. studied the possibility of augmenting SLS algorithms with resolution mechanisms commonly used by complete solvers [9]. There have been fewer studies investigating hybrid approaches to MaxSAT. The most notable work in this area is that of Kroc et al. who proposed a framework where one runs both an SLS solver (WalkSAT) and a complete solver (MiniSAT [6]) in parallel on the same problem instance [14]. During the run, WalkSAT is forbidden from flipping any variable that has already been fixed by MiniSAT, but is otherwise unconstrained. MiniSAT, with its systematic approach, directs the search towards promising regions in the

search space, where WalkSAT is then tasked with quickly configuring the free variables. The authors demonstrate impressive results across a wide range of challenging MaxSAT instances with this technique [14]. The approach, however, fails on instances that are easily proven unsatisfiable, as in these cases, the MiniSAT solver terminates too quickly to be of much assistance to WalkSAT. Our algorithm, on the other hand, can still be applied to such problems and only relies on a single-processor programming model.

Similarly, there have also been several prior attempts to bring the strength of MCTS methods to bear on various combinatorial problems. One of the first such attempts was that of Previti et al. who described a UCT inspired algorithm for solving SAT instances [20]. Since then, others have investigated the feasibility of using MCTS methods for solving mixed integer programming [22] and constraint programming problems [15]. To our knowledge, this is the first work to study the potential of UCT in the MaxSAT setting.

3 The UCTMAXSAT Algorithm

We now describe UCTMAXSAT, a derivative of the original UCT algorithm with specific enhancements for solving unweighted MaxSAT problems. UCTMAXSAT constructs a search tree where each node corresponds to a variable in the supplied Boolean CNF formula φ. Every non-terminal node is treated like a two-armed bandit. Playing the "left" arm (i.e., exploring the subtree rooted at the left child of a node) corresponds to setting a variable to false, while playing the "right" arm corresponds to setting the variable to true. If we denote the set of variables in φ by V, then a sequence of $|V|$ such plays corresponds to a complete truth assignment to the variables. UCTMAXSAT incrementally grows a search tree by iterating over the following steps.

Tree Descent: At a node s, the algorithm chooses whether to go left or right by computing an upper-confidence bound on each child node's estimated value as shown [1]:

$$\text{UCB1}(s_i) = V(s_i) + c \cdot \sqrt{\frac{\ln n(s)}{n(s_i)}} \tag{1}$$

Here, $i \in \{0, 1\}$ refers to the arm index, s_i is the child node reached by playing arm i, $V(s_i)$ is the estimated value of node s_i (and is discussed in greater detail below), $n(s)$ denotes the number of prior visits to the node s, and c is the exploration bias parameter. Starting at the root node, UCTMAXSAT repeatedly selects the arm with the higher UCB1 score (breaking ties arbitrarily) to descend down the currently maintained search tree, until a previously unexpanded node (i.e., one for which $n(s) = 1$) or a terminal node (one corresponding to a complete assignment) is reached. With each variable assignment (arm play), the original formula is appropriately simplified — for convenience, we will use φ' to refer to this simplified formula in contrast to the original formula φ.

Variable Installation: If we have assigned all variables at the conclusion of the tree descent phase, then we compute the fraction of satisfied clauses in the formula and skip ahead to the "Value Propagation" phase. Otherwise, we are left with a partial truth assignment and must decide how to proceed with the current iteration of the search. In particular, we must decide what variable to branch on at the current node. We investigated a number of different heuristics for making this choice, that are summarized in Table 1. Intuitively, these heuristics may be understood as follows:

- $A(k)$: prefers variables that appear in many clauses in φ' — assigning such variables first simplifies the formula faster.
- $S(k)$: prefers variables that appear roughly the same number of times with each sign in φ' — such variables may be better candidates for systematic exploration.
- $M(k)$: prefers variables that both appear in many clauses, and in a "balanced" way, in φ'.

The setting of the parameter k can be used to weight clauses according to their size: $k = -1$ weights short clauses more heavily, $k = 1$ weights long clauses more heavily, and $k = 0$ weights all clauses equally. In addition, we also experimented with other heuristics from the SAT literature including the MOM, Bohm's, Jeroslow-Wang [24] and Max-Falsified [17] heuristics. The results we present in this paper are from using the $A(0)$ heuristic in UCTMAXSAT, owing to its simplicity and robust performance across a range of problem categories.

Tree Growth: Once a variable v has been installed at the current node s, we create the left and right child of this node, that we denote by s_l and s_r respectively. The node s_l is reached by setting v to false at s, while s_r corresponds to setting v to true at s. Both nodes are added to the search tree — thus, each iteration of UCTMAXSAT increases the size of the tree by two nodes.

Value Estimation: Once the tree has been augmented with s_l and s_r, we estimate the value of the decision to install v at s, i.e., the value of further exploring the search subspace rooted at the node s. We do this by performing two SLS runs, one starting at s_l (i.e., with v set to false) and one starting at s_r (with v set to true). We note the similarity between this estimation approach and the idea of *random playouts* that are commonly used

Table 1. The variable selection heuristics used. Here, $C(l)$ denotes the set of clauses in the simplified formula φ' in which the literal l appears, $|c|$ denotes the length of clause c, and $k \in \{-1, 0, 1\}$.

$$A(k): \operatorname{argmax}_l \left[\sum_{c \in C(l)} |c|^k + \sum_{c \in C(\neg l)} |c|^k \right]$$

$$S(k): \operatorname{argmin}_l \left| \sum_{c \in C(l)} |c|^k - \sum_{c \in C(\neg l)} |c|^k \right|$$

$$M(k): \operatorname{argmax}_l \left[\left(\sum_{c \in C(l)} |c|^k \right) \left(\sum_{c \in C(\neg l)} |c|^k \right) \right]$$

in UCT-based game-playing programs [7,8]. However, within our context, a random playout would correspond to simply assigning all free variables arbitrarily; instead, we start with an assignment to the free variables that we then attempt to iteratively improve using a local search. Critically, this initial assignment of the free variables is not random — instead, we assign these variables the same truth values as in the best global solution found by our algorithm so far. Then, during the ensuing SLS runs, we prohibit the algorithm from flipping any variables whose values have been already fixed higher up in the search tree. This is similar to the approach of Kroc et al. [14]. The SLS algorithm is thus constrained to explore the area of the search space that it has been corralled into by the tree descent process. For each SLS run, we keep track of the best solution seen (i.e., the highest fraction of clauses that were satisfied), which we denote by m_l and m_r. We then estimate the value of the node s to be $m = (m_l^2 + m_r^2)/2$. The intuition for why we use m_l^2 and m_r^2, rather than m_l and m_r directly, is the following: on most MaxSAT instances, it is fairly easy to satisfy a large percentage of clauses using any local search procedure, before the search process plateaus. Using a non-linear function of m_l/m_r allows us to magnify the difference between two high-quality candidate solutions. We also experimented with other functional forms — higher degree polynomials and exponentials — but using the square empirically proved to be the best choice for a wide range of problems.

Value Propagation: Once the SLS runs complete (after some fixed number of variable flips), the estimated value of the node s (i.e., m) is propagated up the tree. The value of each node s' that is on the path from the root node to s is updated using a simple averaging operator, as in the original UCT implementation [13]. The visit count is also incremented for each such s', as shown:

$$n(s') \leftarrow n(s') + 1$$
$$V(s') \leftarrow V(s') + \frac{(m - V(s'))}{n(s')}$$

In this way, feedback from the SLS runs influences future expansions of the search tree. Since the values m_l and m_r represent a guaranteed lower bound on the proportion of satisfied clauses, we also experimented with using a max backup operator, that sets the value of a node to the maximum of the incumbent and incoming values. This approach has been found to outperform the traditional averaging operator on problems where the heuristic value estimate is reliable [21,22]. In our setting, however, we found the averaging operator to be superior to the max. A node-closure procedure (similar to that described by Previti et al. [20]) is also built into the value propagation phase. If the node s is a terminal node (i.e., it assigns to the last remaining free variable in φ'), then s is marked "closed" as further visits to this node cannot improve our result. These "closed" flags are propagated upwards through the tree so that tree descents always end at non-closed nodes.

The steps outlined above are repeated until the computational budget allotted to the solver is exhausted. The best solution discovered at any point during the algorithm's run is then reported as the final result. A pseudo-code specification of the algorithm is given in Appendix A.

4   Results

We implemented three flavors of the UCTMAXSAT algorithm that only differ in the SLS algorithm that was employed for estimating the values of leaf nodes: UCTMAXSAT$_W$ (uses WalkSAT [23]), UCTMAXSAT$_N$ (uses Novelty [18]), and UCTMAXSAT$_C$ (uses CCLS [16]). The implementations of WalkSAT and Novelty were adapted from the UBCSAT suite [28], while UCTMAXSAT$_C$ was built on an implementation of CCLS provided by its authors. In the remainder of this section, we present our findings from running experiments on problem instances drawn from the 2015 MaxSAT Evaluation[1] and large random 3SAT formulas. All tests were conducted on an Intel Xeon 3.5 GHz quad-core processor running Linux Ubuntu 14.04, with 16 GB of memory.

4.1   The Exploration-Exploitation Trade-Off

We motivated this work by observing that balancing greed and mobility in local search constituted an exploration-exploitation dilemma that MCTS methods like UCT are well-equipped to handle. The parameter c in equation (1) controls this trade-off — smaller values of c correspond to a greater amount of greed and lead to UCTMAXSAT building highly asymmetric trees that are much deeper in more promising regions of the search space. Higher values of c encourage greater exploration and lead to "bushier" trees. The left panel of Fig. 1 highlights the impact of this parameter's setting on the performance of UCTMAXSAT$_W$. The plot shows the average quality of the solutions (over 50 independent runs of 300 s each) found by the algorithm for the `maxcut-140-630-0.7-1` instance, for various values of c. We note that there is a "sweet spot" for the setting of c that produces the best results, a fact we have confirmed for many different problem instances, as well as for UCTMAXSAT$_N$. This indicates that UCTMAXSAT is indeed effective at reconciling the greed-mobility tension in local search and does so in a sophisticated way — Fig. 1 demonstrates that neither pure-exploitation nor pure-exploration strategies are as successful as a blend of the two.

4.2   Allocating the Computational Budget

In this section, we address the following question: *How much of a fixed computational budget should we allocate to the tree-building and SLS components of* UCTMAXSAT? Using a fixed budget of 300 s of runtime, we investigated various allocation strategies, measured in flips per SLS run. For example, using 1000 flips per SLS run would permit us to run more iterations of UCTMAXSAT (therefore
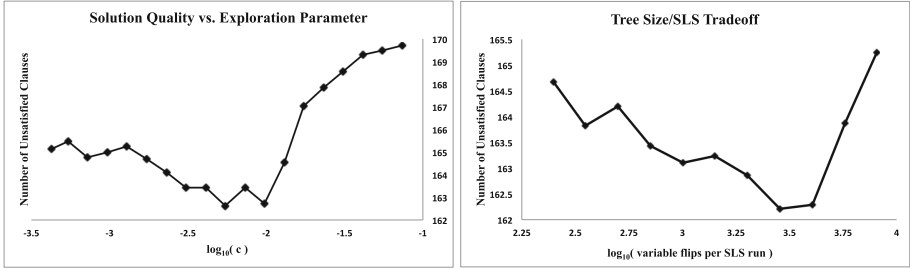

Fig. 1. Left: Effect of the exploration bias parameter c on the quality of the solutions found by UCTMAXSAT$_W$. Right: Comparison of different allocations of the computational budget to the tree-building and SLS components of UCTMAXSAT$_W$.

producing a larger search tree) than using 2000 flips per SLS run (which would require relatively more of the runtime budget to be allocated to the SLS runs). The right panel of Fig. 1 shows the effect of different allocations on the performance of UCTMAXSAT$_W$ on the `maxcut-140-630-0.7-1` problem instance. Each data point represents the average number of unsatisfied clauses found over 50 independent runs. At the left extreme of the plot, we are choosing very short SLS runs that permit us to build larger search trees; at the right extreme, we are performing long SLS runs and building smaller trees. The shape of the curve indicates an optimal allocation that is far removed from both extremes — at this point, there is a synergy between the tree-building and SLS components, and the combination of the two techniques outperforms either component by itself. We found this too to be a robust trend, that persisted independent of the problem instance used, the SLS algorithm or the setting of c. In all the experiments that follow, we use an allocation of 2000 flips per SLS run, which we empirically found to be the best general-purpose setting.

4.3 UCTMAXSAT Against Baseline SLS Algorithms

We now present a head-to-head comparison of two baseline SLS algorithms, WalkSAT and Novelty, to their MCTS-enhanced counterparts. Since the problem instances we use in our experiments vary widely in size, we report our results using the following *normalized difference in solution quality* metric:

$$\sigma(s_1, s_2) = \frac{s_1 - s_2}{\max\{s_1, s_2\}}$$

Here, s_1 and s_2 represent the quality of the best solutions (as measured by the number of unsatisfied clauses) found by the two algorithms being compared. We follow the convention of always using s_1 for the solution found by the SLS algorithm and s_2 for the solution found by UCTMAXSAT. Thus, positive values of σ indicate instances where UCTMAXSAT outperforms its SLS competition, while negative values of σ indicate instances where the situation is reversed. Dividing the difference in solution quality by the max term ensures that σ stays bounded within $[-1, +1]$. However, note that the measure is non-linear: for example, a
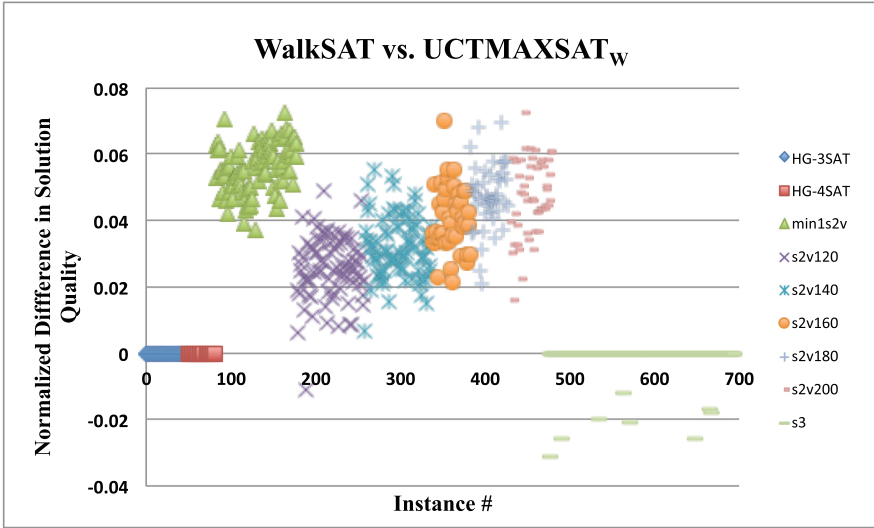
Fig. 2. A comparison of the solutions found by WalkSAT to those found by UCTMAXSAT$_W$ on random instances from the 2015 MaxSAT Evaluation.

score of $\sigma = 0.5$ indicates that UCTMAXSAT found a solution that violates only half as many clauses as the best solution found by its SLS counterpart. A score of $\sigma = 0.9$ indicates that the UCTMAXSAT solution is superior by an order-of-magnitude while scores close to 1.0 denote cases where UCTMAXSAT improved upon the SLS solution by several orders-of-magnitude.

Figures 2 and 3 present our results on the complete set of random unweighted MaxSAT problem instances from the 2015 MaxSAT Evaluation. Each problem instance corresponds to a data point, for which we report the σ measure (as defined earlier). The instances are color-coded according to the problem family from which they are drawn. The critical parameters for the algorithms (noise settings for WalkSAT and Novelty, c for UCTMAXSAT) were tuned on a per-instance basis and we compare the results obtained using the best parameter settings in each case. All results were collected based on a single run of the appropriate algorithm for 300 s — the same time control that is used in the MaxSAT Evaluation. As can be seen from these results, the MCTS tree improves the performance of both vanilla WalkSAT and Novelty on a broad range of instances. In particular, it improves on the best WalkSAT solution (i.e., $\sigma > 0$) on 393 instances and improves on the best Novelty solution on 364 instances (out of 695). There are 287 instances on which WalkSAT and UCTMAXSAT$_W$ tie (i.e., $\sigma = 0$); however, on 286 of these instances, WalkSAT found the optimal solution unaided, thus leaving no room for improvement. There were only 15 instances where UCTMAXSAT$_W$ was outperformed by WalkSAT (i.e., where WalkSAT found a solution with at least one fewer unsatisfied clause than UCTMAXSAT$_W$). Similarly, there was no room for improvement on 293 of the 309 instances where Novelty and UCTMAXSAT$_N$ tie; overall, Novelty outperformed UCTMAXSAT$_N$ on only 22 (out of 695) instances. This performance breakdown on the random instances

Table 2. A breakdown of the difference in performance between the specified SLS algorithm and its MCTS-enhanced counterpart on the random and crafted instances from the 2015 MaxSAT evaluation. The $\sigma > 0$ column presents the number of instances on which the MCTS algorithm outperformed SLS, the $\sigma < 0$ column indicates the number of instances on which the situation was reversed, and the $\sigma = 0$ column gives the number of instances for which both methods found solutions of equal quality.

	Random instances			Crafted instances		
	$\sigma > 0$	$\sigma = 0$	$\sigma < 0$	$\sigma > 0$	$\sigma = 0$	$\sigma < 0$
WalkSAT	393 (56.5 %)	287 (41.3 %)	15 (2.2 %)	336 (83.6 %)	63 (15.7 %)	3 (0.7 %)
Novelty	364 (52.4 %)	309 (44.5 %)	22 (3.1 %)	311 (77.4 %)	76 (18.9 %)	15 (3.7 %)

is summarized in the left section of Table 2. Overall, UCTMAXSAT$_W$ improves on WalkSAT by larger margins on average than UCTMAXSAT$_N$ improves on Novelty — this is unsurprising given that Novelty is a superior SLS algorithm and is thus more challenging to surpass.

Figures 4 and 5 present our results on the complete set of crafted unweighted MaxSAT instances from the 2015 MaxSAT evaluation. Overall, the trend is similar to what was observed on the random instances: once again, UCTMAXSAT outperforms WalkSAT and Novelty across a wide variety of benchmark domains, it tends to improve WalkSAT more so than Novelty, and in the overwhelming majority of cases when the algorithms tie, there is no room for improvement since both algorithms find the optimal solution. Interestingly, looking across the left and right partitions of Table 2, we see that the impact of MCTS on the underlying SLS algorithm is more pronounced on crafted instances than
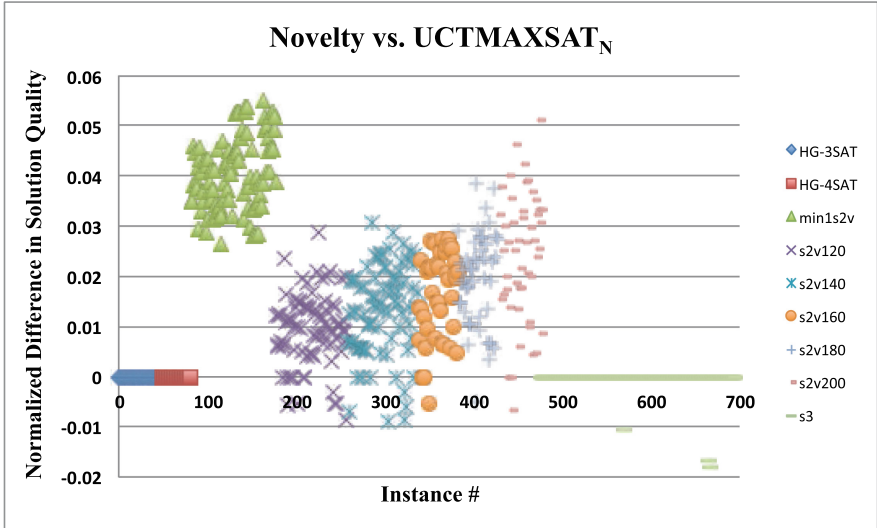

Fig. 3. A comparison of the solutions found by Novelty to those found by UCTMAXSAT$_N$ on random instances from the 2015 MaxSAT Evaluation.
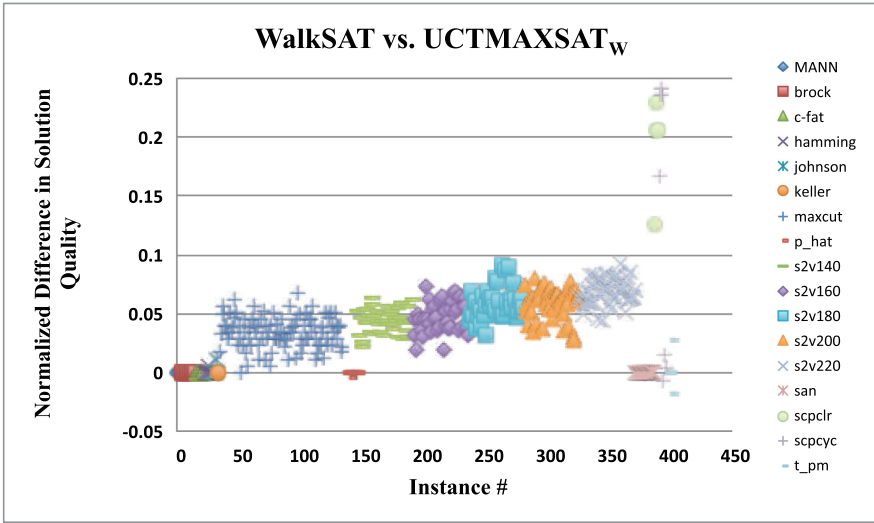
Fig. 4. A comparison of the solutions found by WalkSAT to those found by UCTMAXSAT$_W$ on crafted instances from the 2015 MaxSAT Evaluation.
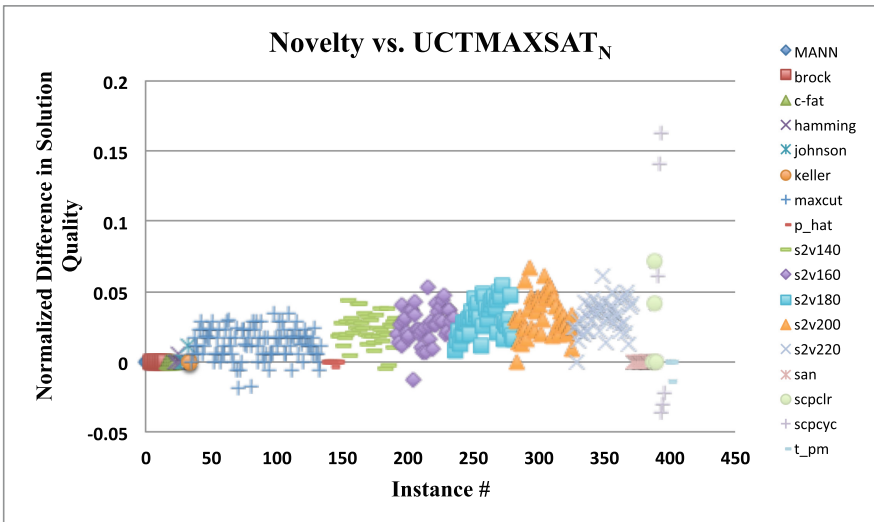

Fig. 5. A comparison of the solutions found by Novelty to those found by UCTMAXSAT$_N$ on crafted instances from the 2015 MaxSAT Evaluation.

on random instances. This is consistent with results that are well-known in the SAT community, namely that systematic approaches fare better in domains with more structure, whereas local search approaches perform better on random formulas [11].
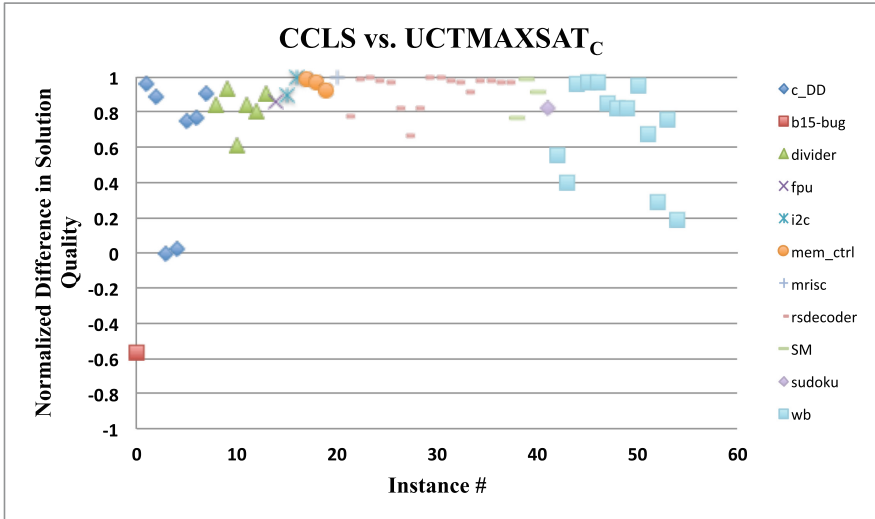
Fig. 6. A comparison of the solutions found by CCLS to those found by UCTMAXSAT$_C$ on industrial instances from the 2015 MaxSAT Evaluation.eps

4.4 Uctmaxsat$_C$ Against CCLS

In this section, we consider the benefits of augmenting CCLS [16], the top-performing incomplete MaxSAT solver in the Random and Crafted categories from the 2015 MaxSAT Evaluation[2], with a UCT tree search. We focused our initial attention on instances from the Industrial track of the MaxSAT Evaluation, having observed that CCLS fared poorly in this category. Using the same experimental set-up as in Sect. 4.3 (a single run of 300 s using the best parameter setting for each algorithm), we obtained the results shown in Fig. 6. As can be seen, UCTMAXSAT$_C$ handsomely outperforms CCLS on all but three instances. Indeed, the margin of improvement on many of the instances is by several orders of magnitude. A particularly stark example is the `mrisc_mem2wire` instance (the blue-grey '+' data point in the plot) where the best solution discovered by CCLS violates 397,032 clauses. In contrast, the best solution found by UCTMAXSAT$_C$ only violates 386 clauses. We saw similar margins of improvement on many of the other instances as well.

We also conducted experiments comparing the performance of UCTMAXSAT$_C$ to CCLS on random 3SAT formulas. Given that CCLS found the optimal solution for every single random instance that was used in the 2015 MaxSAT Evaluation, we chose to work with more challenging instances instead. We generated random 3SAT formulas with 700, 900, and 1100 variables, while maintaining the same clause-to-variable ratios as those used in the Evaluation. Figure 7 summarizes our results over a set of 105 formulas. CCLS outperforms UCTMAXSAT$_C$ on all

---

[2] http://www.maxsat.udl.cat/15/results-incomplete/index.html, retrieved on Feb. 21, 2016.
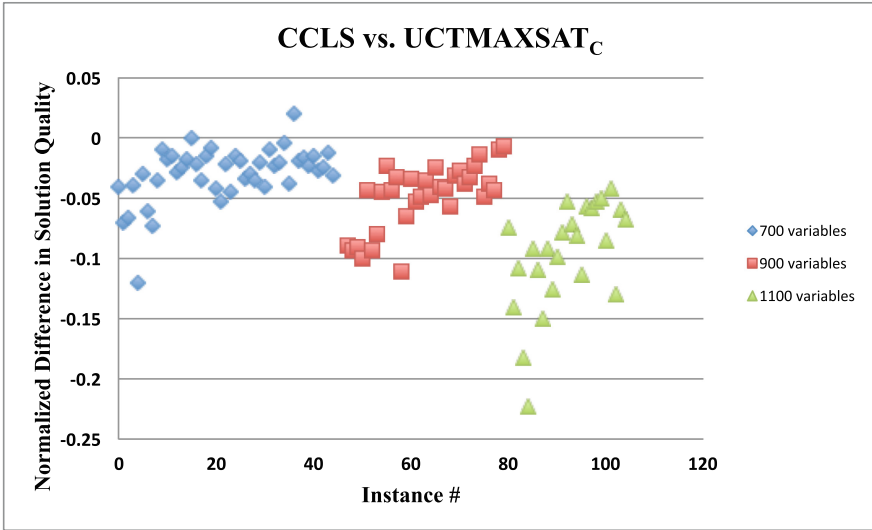
Fig. 7. A comparison of the solutions found by CCLS to those found by UCTMAXSAT$_C$ on large random 3SAT instances.

but two instances in this setting. It appears that the systematic tree search component really improves the performance of the underlying SLS solver on structured instances, but offers little benefit on random instances.

4.5 Summary

In summary, overlaying an MCTS search tree on top of existing SLS solvers has the effect of making them more "well-rounded" — the underlying solvers are improved in their traditional areas of weakness. For example, it is well-known that solvers like WalkSAT perform poorly on overconstrained random MaxSAT instances [29]. UCTMAXSAT$_W$ is able to boost the performance of native WalkSAT in this domain. Similarly, while CCLS is a top-performing solver on random MaxSAT instances, industrial problems are its Achilles Heel. In this case, UCTMAXSAT$_C$ substantially improves the performance of CCLS on industrial instances.

5 Future Work

This paper presented some exploratory work that demonstrated how techniques borrowed from the MCTS framework could improve incomplete solution methods for MaxSAT problems. We offer a few possible directions for future work.

– There is a large region of the algorithm parameter space that is ripe for exploration — for example, one could consider alternatives to the UCB1 bandit algorithm such as ϵ_n-GREEDY [1], or test other value back up operators (like a soft-max).

– In this work, we started with established SLS algorithms and attempted to "boost" their performance by combining them with tree search. However, many in the game-playing community have observed that certain random playout strategies work better in conjunction with UCT than others [8,25]. It would be interesting to investigate whether one could design novel SLS algorithms that specifically dovetail well with UCTMAXSAT (but are not necessarily effective as stand-alone solvers).

– Finally, it would be interesting to look into how one could extend the UCTMAXSAT algorithm to handle partial and weighted MaxSAT instances, and how the approach compares to the current state-of-the-art solvers.

6   Conclusions

In this paper, we presented a novel Monte-Carlo Tree Search based approach for solving MaxSAT problems. Our algorithm, UCTMAXSAT, elegantly combines aspects of systematic and local search. Our experiments with problem instances from the 2015 MaxSAT Evaluation show that UCTMAXSAT significantly improves the performance of baseline incomplete algorithms like WalkSAT and Novelty on almost every problem domain. Our algorithm also improves the performance of CCLS, a state-of-the-art incomplete solver, on industrial instances. In the future, we plan to expand on this work to extract better performance across a wider range of MaxSAT problem instances.

A   Appendix: The Uctmaxsat Algorithm

Algorithm 1. UCTMAXSAT main loop

1:
2: **globals:**
3:   φ: the input (non-empty) CNF formula
4:   $\rho_{best} \leftarrow$ a random complete assignment   ▷ best solution found so far
5:
6: **function** UCTMAXSAT
7:     $root \leftarrow$ CREATENODE($\emptyset$)   ▷ pointer to root node of search tree
8:     **repeat**
9:         DESCEND($root$, φ, $\emptyset$)
10:     **until** time limit is met
11:     **return** ρ_{best}
12: **end function**
13:

Algorithm 2. UCTMAXSAT helper routines

1:
2: **function** CREATENODE(ρ)
3:       $node \leftarrow$ new node
4:       $node.visits \leftarrow 1$
5:       $\rho' \leftarrow$ set of literals in ρ_{best} that do not contradict a literal in ρ
6:       Run SLS algorithm using $\rho \cup \rho'$ as starting configuration, never flipping variables in ρ
7:       $f \leftarrow$ fraction of satisfied clauses in best assignment found by SLS run
8:       $node.value \leftarrow f^2$
9:       Update ρ_{best} if the SLS run uncovered a new best solution
10:       **return** $node$
11: **end function**
12:
13: **function** DESCEND($node$, φ', ρ)   ▷ ρ is the current partial assignment (set of literals)
14:       **if** ρ is a complete assignment **then**
15:             Mark $node$ as closed
16:             $f \leftarrow$ fraction of clauses in φ that are satisfied by ρ
17:             **return** f^2
18:       **else if** $node.visits = 1$ **then**   ▷ a previously unexpanded node
19:             $node.variable \leftarrow A(0)$ ▷ (see Table 1)
20:             $node.left \leftarrow$ CREATENODE($\rho \cup \{\neg node.variable\}$)
21:             $node.right \leftarrow$ CREATENODE($\rho \cup \{node.variable\}$)
22:             $r \leftarrow (node.left.value + node.right.value)/2$
23:       **else**
24:             $child, \varphi', \rho \leftarrow$ SELECTCHILD($node$, φ', ρ)
25:             $r \leftarrow$ DESCEND($child$, φ', ρ)
26:       **end if**
27:       $node.visits \leftarrow node.visits + 1$
28:       $node.value \leftarrow node.value + (r - node.value)/node.visits$
29:       Mark $node$ as closed if $node.left$ and $node.right$ are both closed
30:       **return** v
31: **end function**
32:
33: **function** SELECTCHILD($node$, φ', ρ)
34:       **if** either $node.left$ or $node.right$ is closed **then**
35:             $ch \leftarrow$ the still open child
36:       **else**
37:             $ch \leftarrow$ child that maximizes UCB1 score   ▷ (see Eq. 1)
38:       **end if**
39:       **if** ch is the left child of $node$ **then**
40:             $\varphi' \leftarrow \varphi'$ simplified by assigning $node.variable$ to FALSE
41:             $\rho \leftarrow \rho \cup \{\neg node.variable\}$
42:       **else**
43:             $\varphi' \leftarrow \varphi'$ simplified by assigning $node.variable$ to TRUE
44:             $\rho \leftarrow \rho \cup \{node.variable\}$
45:       **end if**
46:       **return** (ch, φ', ρ)
47: **end function**
48:

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**(2–3), 235–256 (2002). doi:10.1023/A:1013689704352
2. Balla, R.K., Fern, A.: UCT for tactical assault planning in real-time strategy games. In: Proceedings of the 21st International Joint Conference on Artifical Intelligence, IJCAI 2009, pp. 40–45. Morgan Kaufmann Publishers Inc., San Francisco (2009). http://dl.acm.org/citation.cfm?id=1661445.1661453
3. Branavan, S.R.K., Silver, D., Barzilay, R.: Learning to win by reading manuals in a Monte-Carlo framework. J. Artif. Intell. Res. (JAIR) **43**, 661–704 (2012). doi:10.1613/jair.3484
4. Cai, S., Su, K.: Local search for boolean satisfiability with configuration checking and subscore. Artif. Intell. **204**, 75–98 (2013). doi:10.1016/j.artint.2013.09.001
5. Ciancarini, P., Favini, G.P.: Monte carlo tree search in Kriegspiel. Artif. Intell. **174**(11), 670–684 (2010). http://www.sciencedirect.com/science/article/pii/S0004370210000536
6. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
7. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: Proceedings of the 23rd National Conference on Artificial Intelligence, vol. 1, AAAI 2008, pp. 259–264. AAAI Press (2008). http://dl.acm.org/citation.cfm?id=1619995.1620038
8. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Proceedings of the 24th International Conference on Machine Learning, ICML 2007, NY, USA, pp. 273–280 (2007). http://doi.acm.org/10.1145/1273496.1273531
9. Habet, D., Li, C.-M., Devendeville, L., Vasquez, M.: A hybrid approach for SAT. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 172–184. Springer, Heidelberg (2002)
10. Hoos, H., Sttzle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann Publishers Inc., San Francisco (2004)
11. H. Hoos, H., Stützle, T.: Systematic vs. local search for SAT. In: Burgard, W., Christaller, T., Cremers, A.B. (eds.) KI 1999. LNCS (LNAI), vol. 1701, pp. 289–293. Springer, Heidelberg (1999)
12. Keller, T., Eyerich, P.: PROST: probabilistic planning based on UCT. In: McCluskey, L., Williams, B., Silva, J.R., Bonet, B. (eds.) Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, 25–29 June 2012. AAAI (2012). http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4715
13. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
14. Kroc, L., Sabharwal, A., Gomes, C.P., Selman, B.: Integrating systematic and local search paradigms: a new strategy for maxsat. In: Boutilier, C. (ed.) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, 11–17 July 2009, pp. 544–551 (2009) http://ijcai.org/papers09/Papers/IJCAI09-097.pdf
15. Loth, M., Sebag, M., Hamadi, Y., Schoenauer, M.: Bandit-based search for constraint programming. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 464–480. Springer, Heidelberg (2013)

16. Luo, C., Cai, S., Wu, W., Jie, Z., Su, K.: CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans. Comput. **64**(7), 1830–1843 (2015)

17. Mazure, B., Saïs, L., Grégoire, E.: Boosting complete techniques thanks to local search methods. Ann. Math. Artif. Intell. **22**(3–4), 319–331 (1998). doi:10.1023/A:1018999721141

18. McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence, AAAI 1997/IAAI 1997, pp. 321–326. AAAI Press (1997). http://dl.acm.org/citation.cfm?id=1867406.1867456

19. Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and core-guided maxsat solving: a survey and assessment. Constraints **18**(4), 478–534 (2013)

20. Previti, A., Ramanujan, R., Schaerf, M., Selman, B.: Applying UCT to boolean satisfiability. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 373–374. Springer, Heidelberg (2011)

21. Ramanujan, R., Selman, B.: Trade-offs in sampling-based adversarial planning. In: Bacchus, F., Domshlak, C., Edelkamp, S., Helmert, M. (eds.) Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany, 11–16 June 2011. AAAI (2011). http://aaai.org/ocs/index.php/ICAPS/ICAPS11/paper/view/2708

22. Sabharwal, A., Samulowitz, H., Reddy, C.: Guiding combinatorial optimization with UCT. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 356–361. Springer, Heidelberg (2012)

23. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Hayes-Roth, B., Korf, R.E. (eds.) Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, 31 July - 4 August 1994, vol. 1, pp. 337–343. AAAI Press/The MIT Press (1994). http://www.aaai.org/Library/AAAI/1994/aaai94-051.php

24. Marques-Silva, J.: The impact of branching heuristics in propositional satisfiability algorithms. In: Barahona, P., Alferes, J.J. (eds.) EPIA 1999. LNCS (LNAI), vol. 1695, pp. 62–74. Springer, Heidelberg (1999)

25. Silver, D., Tesauro, G.: Monte-carlo simulation balancing. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, NY, USA, pp. 945–952 (2009). http://doi.acm.org/10.1145/1553374.1553495

26. Silver, D., Veness, J.: Monte-Carlo planning in large POMDPS. In: Lafferty, J.D., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., Culotta, A. (eds.) Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010, Proceedings of a Meeting Held 6–9 December 2010, Vancouver, British Columbia, Canada, pp. 2164–2172. Curran Associates, Inc. (2010). http://papers.nips.cc/paper/4031-monte-carlo-planning-in-large-pomdps

27. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning, 1st edn. MIT Press, Cambridge (1998)

28. Tompkins, D.A.D., Hoos, H.H.: UBCSAT: an implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10–13 May 2004, Vancouver, BC, Canada, Online Proceedings (2004). http://www.satisfiability.org/SAT04/programme/23.pdf

29. Zhang, W., Rangan, A., Looks, M.: Backbone guided local search for maximum satisfiability. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI 2003, pp. 1179–1184. Morgan Kaufmann Publishers Inc., San Francisco (2003)

A New Approach to Checking the Dynamic Consistency of Conditional Simple Temporal Networks

Luke Hunsberger[1]([✉]) and Roberto Posenato[2]

[1] Vassar College, Poughkeepsie, NY 12604, USA
hunsberger@vassar.edu
[2] University of Verona, Verona, Italy
roberto.posenato@univr.it

Abstract. A Conditional Simple Temporal Network (CSTN) is a structure for representing and reasoning about temporal constraints in domains where constraints may apply only in certain scenarios. Observations in real time incrementally reveal the "real" scenario. A CSTN is *dynamically consistent* (DC) if there is a strategy for executing its time-points that guarantees the satisfaction of all relevant constraints. The fastest DC-checking algorithm for CSTNs is based on constraint propagation. This paper introduces a new approach to DC checking for CSTNs, inspired by controller-synthesis algorithms for Timed Game Automata. The new algorithm views the DC-checking problem as a two-player game, searching an abstract game tree to find a "winning" strategy, using Monte-Carlo Tree Search and Limited Discrepancy Search to guide its search. An empirical evaluation shows that the new algorithm is competitive with the propagation-based algorithm.

1 Introduction

Recently, there have been significant advances in the theory and practice of temporal networks and Timed Game Automata (TGAs). Of particular interest is the work by Cimatti et al. [4,5] which showed that dynamic consistency/controllability (a.k.a., DC-checking) problems for a variety of temporal networks can be reduced to controller-synthesis problems for TGAs. Although their work revealed strong theoretical connections between temporal networks and TGAs, it did not immediately result in practical algorithms because the generic TGA solver (UPPAAL-TIGA [2,3]) could not exploit the particular structure of the DC-checking problem. However, more recently, Cimatti et al. [6] presented a practical DC-checking algorithm for one kind of network—a Disjunctive Temporal Network with Uncertainty (DTNU)—by using the same temporal-network-to-TGA reduction, but customizing the controller-synthesis algorithm to exploit the particular structure of the DC-checking problem for DTNUs.

Inspired by their approach, this paper presents a new DC-checking algorithm for a different kind of network: a Conditional Simple Temporal Network (CSTN).

Although the spirit of the approach is similar, especially in the traversal of an abstract *simulation graph,* the very different execution semantics for CSTNs and DTNUs required the development of numerous novel representations and algorithmic techniques. Notably, the new algorithm does not reduce CSTNs to TGAs; instead, it maps TGA-based techniques into the realm of temporal networks, using basic consistency-checking algorithms for Simple Temporal Networks (STNs) [8], whereas Cimatti et al. use techniques from Satisfiability Modulo Theory (SMT) [1]. In addition, the new algorithm uses Monte-Carlo Tree Search (MCTS) [9] and Limited Discrepancy Search (LDS) [10] to guide its search. An empirical evaluation demonstrates that the new algorithm is competitive with the fastest known DC-checking algorithm for CSTNs from earlier work, which is based on a very different approach [13]. Although the earlier algorithm can be faster for weakly constrained networks and inconsistent networks, the two algorithms perform similarly for moderately constrained networks. In addition, the new algorithm efficiently handles certain worst-case structures that dramatically slow down the earlier algorithm.

2   Background

A *Simple Temporal Network* (STN) is a structure for representing and reasoning about time [8]. An STN is a pair $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set of real-valued variables called time-points (notated $X, Y, Z, \ldots$), and $\mathcal{C}$ is a set of binary difference constraints on those time-points (e.g., $Y - X \leq 5$). The graph for an STN is a pair $(\mathcal{T}, \mathcal{E})$, where each constraint, $Y - X \leq \delta$ in $\mathcal{C}$, corresponds to an edge in $\mathcal{E}$ from X to Y of length δ. An STN is *consistent* if it has a solution as a constraint satisfaction problem. The consistency-checking problem for STNs can be solved in cubic time by computing the corresponding *all-pairs, shortest-paths* (APSP) matrix—called its *distance matrix* $\mathcal{D}$. For each $X, Y \in \mathcal{T}$, $\mathcal{D}(X, Y)$ equals the length of the shortest path from X to Y.

Theorem 1 (Decomposability of STNs [8,11]). *Let $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ be any STN, and $\mathcal{D}$ its distance matrix. For any $\mathcal{X} \subseteq \mathcal{T}$, let $\mathcal{C}|_{\mathcal{X}} = \{(Y - X \leq \mathcal{D}(X, Y)) \mid X, Y \in \mathcal{X}\}$ be the shortest-path constraints for paths in $\mathcal{S}$ whose end-points are in $\mathcal{X}$. Then any solution for the STN $\mathcal{S}|_{\mathcal{X}} = (\mathcal{X}, \mathcal{C}|_{\mathcal{X}})$ can be extended to form a solution for $\mathcal{S}$; and the distance matrix $\mathcal{D}|_{\mathcal{X}}$ for $\mathcal{S}|_{\mathcal{X}}$ satisfies: for all $X, Y \in \mathcal{X}$, $\mathcal{D}|_{\mathcal{X}}(X, Y) = \mathcal{D}(X, Y)$. (For convenience, $\mathcal{D}|_{\mathcal{X}}$ may be called the* restriction *of $\mathcal{D}$ to time-points in $\mathcal{X}$.)*

Lemma 1 (Intersecting STN solution sets). *Let $\mathcal{S}_1 = (\mathcal{T}, \mathcal{C}_1)$ and $\mathcal{S}_2 = (\mathcal{T}, \mathcal{C}_2)$ be STNs over the same set of time-points, $\mathcal{T}$; and let $\mathcal{D}_1$ and $\mathcal{D}_2$ be the corresponding distance matrices. Next, let $\mathcal{D}'$ be the* element-wise minimum *of the matrices $\mathcal{D}_1$ and $\mathcal{D}_2$; and let $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ be the STN where $\mathcal{C} = \{(Y - X \leq \mathcal{D}'(X, Y))\}$. Then the solution set for $\mathcal{S}$ is the intersection of the solution sets for $\mathcal{S}_1$ and $\mathcal{S}_2$.*

Proof. Let σ be a solution for both $\mathcal{S}_1$ and $\mathcal{S}_2$. Let $X, Y \in \mathcal{T}$ be arbitrary. Since σ is a solution for $\mathcal{S}_1$, $\sigma(Y) - \sigma(X) \leq \mathcal{D}_1(X, Y)$; and since σ is a solution for $\mathcal{S}_2$, $\sigma(Y) - \sigma(X) \leq \mathcal{D}_2(X, Y)$. Thus, $\sigma(Y) - \sigma(X) \leq \mathcal{D}'(X, Y)$, implying that σ is a solution for $\mathcal{S}$. The opposite direction follows similarly.   □

The distance matrix $\mathcal{D}$ for $\mathcal{S}$ may be called the *intersection* of $\mathcal{D}_1$ and $\mathcal{D}_2$, notated as $\mathcal{D} = \mathcal{D}_1 \cap \mathcal{D}_2$. However, since taking the element-wise minimum may introduce new shortest paths, entries of $\mathcal{D}$ may be *less than* the corresponding entries of $\mathcal{D}_1$ and $\mathcal{D}_2$.

Extending Expressiveness. The expressiveness of an STN can be extended in several dimensions: (C) allowing *conditional* constraints; (D) allowing *disjunctive* constraints; and (U) allowing temporal intervals with *uncertain* durations. Different combinations of these features result in networks with names such as *Conditional Simple Temporal Networks* (CSTNs) and *Disjunctive Temporal Networks with Uncertainty* (DTNUs). (For networks that allow disjunctive constraints, the "simple" modifier is dropped.) Arranging these networks according to their expressiveness leads to the hierarchy in Fig. 1.
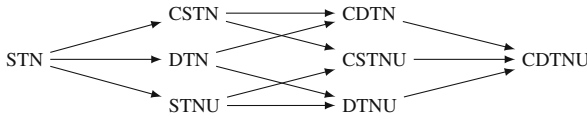

Fig. 1. A hierarchy of temporal networks from least to most expressive

Dynamic Consistency/Controllability. Although the presence of disjunctive constraints makes the consistency-checking problem for Disjunctive Temporal Networks (DTNs) NP-hard [8], the execution semantics for DTNs is the same as that for STNs. In contrast, the presence of conditional constraints (C) or intervals with uncertain durations (U) dramatically changes the execution semantics and, therefore, requires new notions of consistency (or controllability). For example, a CSTN has conditional constraints that may apply only in certain scenarios; and the "real" scenario is only incrementally revealed by the execution of *observation time-points.* A CSTN is *dynamically consistent* if there exists a *dynamic strategy* for executing its time-points such that all relevant constraints will be satisfied no matter which scenario is incrementally revealed [18]. On the other hand, a Simple Temporal Network with Uncertainty (STNU) includes intervals with uncertain durations, represented by *contingent links.* The ending time-points of contingent links are *uncontrollable,* but guaranteed to fall within certain bounds. An STNU is *dynamically controllable* if there exists a dynamic strategy for executing its controllable time-points such that all constraints will be satisfied no matter how the uncertain durations turn out [17]. Finally, the dynamic controllability of CSTNUs and CDTNUs, which allow conditional constraints *and* contingent links, has also been defined [4,12]. (When including both conditional

constraints and contingent links, the term "controllability" is preferred.) Crucially, the decisions made by *dynamic* execution strategies must only depend on past information, whether gleaned from the execution of observation time-points or the observed durations of contingent links.

A *DC-checking algorithm* is an algorithm for checking the dynamic consistency or controllability of a temporal network. DC-checking for STNUs can be done in cubic time [16]. However, consistency checking for DTNs and the DC-checking problem for all other network classes in Fig. 1 (other than STNs) are known to be NP-hard.[1]

Cimatti et al. [4] showed that the DC-checking problem for CDTNUs, the most expressive network in Fig. 1, can be reduced to a controller-synthesis problem for TGAs, but the resulting algorithm was not practical because the generic controller-synthesis algorithm for TGAs could not exploit the DC-checking problem structure. More recently, Cimatti et al. [6] presented a practical DC-checking algorithm for DTNUs, using the same temporal-network-to-TGA reduction, but implementing a customized controller-synthesis algorithm that exploits the structure of the DC-checking problem for DTNUs.

Inspired by their approach, this paper presents a new DC-checking algorithm for CSTNs. Although similar in spirit, the CSTN algorithm differs substantially from the DTNU algorithm: (1) instead of translating CSTNs into TGAs, all computations are done on related STNs; (2) the transitions in the *simulation graph* are *very* different owing to the different execution semantics for CSTNs; (3) the *winning regions* are represented by unions of STNs, not logic-based formulas; and (4) Monte-Carlo Tree Search (MCTS) [9] and Limited Discrepancy Search (LDS) [10] are used to guide its search.

3   Conditional Simple Temporal Networks

This section reviews the dynamic consistency of CSTNs [18], using definitions drawn from Hunsberger et al. [13]. To begin, let $\mathcal{P}$ be a set of propositional letters. A *label* is any consistent conjunction of (positive or negative) literals from $\mathcal{P}$; the set of all such labels is denoted by $\mathcal{P}^*$; and the empty label is denoted by $\Box \in \mathcal{P}^*$.

A CSTN may include time-points and temporal constraints that apply only in certain scenarios. The "real" scenario is not known in advance, but is incrementally revealed through the execution of *observation time-points* (ObsTPs). (For convenience, non-observation time-points may be called *ordinary* time-points (OrdTPs).) Each observation time-point P? has a corresponding propositional letter p; executing P? generates a truth value for p. During execution, the observations that have been made so far are recorded in a label called the *current partial scenario* (CPS). For example, if p and s were observed to be *true*, and q *false*, then the CPS would be the label $p\neg qs$. Time-points and constraints in a

---

[1] Dechter et al. [8] for DTNs; Comin and Rizzi [7] for CSTNs; the rest follow from these results.

CSTN may have propositional labels. For example, $(Y - X \leq 5, p\neg q)$ specifies that $Y - X \leq 5$ must hold in any scenario that is consistent with $p\neg q$.

Definition 1 (CSTN). *A* Conditional Simple Temporal Network *(CSTN) is a tuple,* $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, \mathcal{P} \rangle$, *where:*

- $\mathcal{P}$ *is a finite set of propositional letters;*
- $\mathcal{T}$ *is a finite set of real-valued variables (time-points);*
- $\mathcal{C}$ *is a finite set of labeled constraints, each having the form,* $(Y - X \leq \delta, \ell)$, *where* $X, Y \in \mathcal{T}$, $\delta \in \mathbb{R}$, *and* $\ell \in \mathcal{P}^*$;
- $L : \mathcal{T} \to \mathcal{P}^*$ *is a function assigning labels to time-points;*
- $\mathcal{OT} \subseteq \mathcal{T}$ *is a set of observation time-points (ObsTPs); and*
- $\mathcal{O} : \mathcal{P} \to \mathcal{OT}$ *is a bijection between ObsTPs and propositional letters.*

A CSTN typically includes a special time-point Z whose value is fixed at 0; all other time-points are constrained to occur at or after Z. (If no such Z exists, one may be added without adverse effects.) Binary constraints involving Z are equivalent to unary constraints. For example, $Z - X \leq -5$ is equivalent to $X \geq 5$.

Each CSTN $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \ldots \rangle$, has an associated graph, $\langle \mathcal{T}, \mathcal{E} \rangle$, where the edges in $\mathcal{E}$ correspond to the labeled constraints in $\mathcal{C}$. In particular, each $(Y - X \leq \delta, \ell) \in \mathcal{C}$ corresponds to an edge from X to Y annotated by the *labeled value* $\langle \delta, \ell \rangle$. A sample CSTN graph is shown in Fig. 2. This graph includes structures, called *negative q-loops,* that can dramatically slow down the DC-checking algorithm from earlier work [13].

The execution semantics for a CSTN can be expressed in terms of a two-player game between the agent responsible for executing time-points and the environment responsible for selecting truth values for propositional letters [4]. An execution *run* begins with $Z = 0$ and an empty current partial scenario. At any time, the agent may choose to execute any time-point whose label is entailed by the CPS. Whenever an observation time-point $P?$ is executed, the environment instantaneously selects a truth value for the corresponding letter p. If $p = true$, then p is conjoined to the CPS; otherwise, $\neg p$ is conjoined to the CPS. The execution run is completed whenever it happens that all time-points whose labels are entailed by the CPS have been executed. If all constraints whose labels are consistent with the final CPS are satisfied, then the agent wins; otherwise, the environment wins. A sample winning run for the CSTN from Fig. 2 is given below. (The choices made by the environment are parenthesized.) For this run, the final CPS is $\neg pq\neg w$; thus, constraints labeled by pw, qw and $\neg q\neg w$ need not be satisfied.

$$Z = 0; W? = 100 \ (\neg w); X = 101; Y = 106; P? = 107 \ (\neg p); Q? = 125 \ (q).$$

Definition 2 (Scenario). *A (complete)* scenario *over a set* $\mathcal{P}$ *of propositional letters is a (complete) function,* $s : \mathcal{P} \to \{true, false\}$. *For each label* $\ell \in \mathcal{P}^*$, *the truth value for* ℓ *determined by* s *is denoted by* $s(\ell)$. *The set of all scenarios over* $\mathcal{P}$ *is denoted by* $\mathcal{I}$. *A* partial scenario *is any function,* $s : \mathcal{P}' \to \{true, false\}$, *where* $\mathcal{P}' \subseteq \mathcal{P}$.
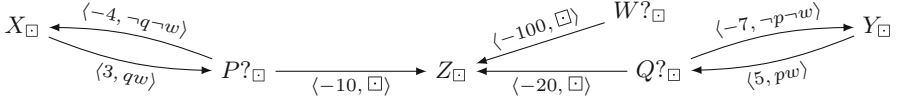
Fig. 2. A CSTN with negative q-loops

Definition 3 (Schedule). *A* schedule *for a set of time-points $\mathcal{T}$ is a (complete) mapping, $\psi : \mathcal{T} \to \mathbb{R}$. The set of all schedules over all subsets of $\mathcal{T}$ is denoted by Ψ.*

Definition 4 (Projection). *Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, \ldots \rangle$ be any CSTN, and s any scenario. The* projection *of $\mathcal{S}$ onto s, notated $\mathcal{S}(s)$, is the STN, $(\mathcal{T}_s^+, \mathcal{C}_s^+)$, consisting of the time-points and constraints whose labels are entailed by s. Thus, $\mathcal{T}_s^+ = \{T \in \mathcal{T} \mid s(L(T)) = true\}$; and $\mathcal{C}_s^+ = \{(Y - X \leq \delta) \mid$ for some $\ell, (Y - X \leq \delta, \ell) \in \mathcal{C}$ and $s(\ell) = true\}$.*

Definition 5 (Execution Strategy). *An* execution strategy *for a CSTN $\mathcal{S}$ is a mapping $\sigma : \mathcal{I} \to \Psi$, such that for each scenario $s \in \mathcal{I}$, the domain of $\sigma(s)$ is $\mathcal{T}_s^+$. If, in addition, for each scenario s, the schedule $\sigma(s)$ is a solution to the projection $\mathcal{S}(s)$, then σ is called* viable. *The execution time for any X in $\sigma(s)$ is denoted by $[\sigma(s)]_X$.*

Definition 6 (History). *Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \ldots \rangle$ be any CSTN, s any scenario, σ any execution strategy for $\mathcal{S}$, and t any real number. The* history *of t in the scenario s, for the strategy σ—notated $Hist(t, s, \sigma)$—is the set of observations made before time t according to $\sigma(s)$: $Hist(t, s, \sigma) = \{(p, s(p)) \mid P? \in \mathcal{T}_s^+$ and $[\sigma(s)]_{P?} < t\}$.*

Definition 7 (Dynamic Execution Strategy). *An execution strategy σ for a CSTN is called* dynamic *if for any scenarios s_1 and s_2, and any time-point X:*

$$\text{let: } t = [\sigma(s_1)]_X; \quad \text{if: } Hist(t, s_1, \sigma) = Hist(t, s_2, \sigma); \quad \text{then: } [\sigma(s_2)]_X = t.$$

The above requirement forces execution decisions to depend only on past observations.

Definition 8 (Dynamic Consistency). *A CSTN $\mathcal{S}$ is* dynamically consistent *(DC) if there exists an execution strategy for $\mathcal{S}$ that is both dynamic and viable.*

4 A New Approach to DC Checking for CSTNs

We view the execution semantics of a CSTN as a two-player game between an agent who executes time-points and the environment that assigns truth values to propositions [4]. Each *run* of the game consists of a sequence of turns. Since the environment is idle until the agent executes an observation time-point, we model an agent's turn as involving the (typically not simultaneous) execution of zero

or more ordinary time-points followed by exactly one observation time-point P?
A turn for the environment involves instantaneously assigning a truth value to p,
the propositional letter associated with P?, resulting in either p or $\neg p$ being
appended to the current partial scenario. The run ends when all time-points
whose labels are entailed by the CPS have been executed. The agent wins if all
constraints entailed by the final CPS are satisfied. The agent seeks a *winning*
strategy (i.e., one that guarantees that all relevant constraints will be satisfied no
matter which choices the environment makes along the way). A *dynamic* strategy
is able to react to past observations. However, to simplify the presentation, this
paper presumes that an agent is also able to react *instantaneously* to observations
(i.e., after zero delay).[2]

Since n time-points can be executed in $n!$ orders, and each time-point can
typically be assigned one of uncountably many values, we use a more abstract
representation for agent moves, one that does not specify execution times for
the time-points being "played". Each (abstract) move is represented by a pair
$(\chi, P?)$, where χ is a possibly empty set of ordinary time-points, and P? is an
observation time-point. For maximal flexibility, the time-points in $\chi \cup P$? are
only partially ordered: for each $X \in \chi$, and each as-yet-unplayed $Y \notin \chi \cup P$?,
the ordering constraints, $X \leq P? \leq Y$, are posted. Thus, a typical sequence of
alternating moves has the following form:

$$(\chi_1, P_1?), (p_1 = b_1), (\chi_2, P_2?), (p_2 = b_2), \ldots, (\chi_k, P_k?), (p_k = b_k), \chi_{k+1}$$

where the χ_i are disjoint sets of ordinary time-points, the P_i? are distinct obser-
vation time-points, the p_i are the corresponding propositional letters, the b_i are
truth values, and the last agent move (i.e., χ_{k+1}) involves only ordinary time-
points. The associated ordering constraints can be concisely expressed as follows:

$$Z = 0 \leq \chi_1 \leq P_1? \leq \chi_2 \leq P_2? \leq \ldots \leq \chi_k \leq P_k? \leq \chi_{k+1}$$

where expressions of the form, $\chi_i \leq P_i? \leq \chi_{i+1}$, stand for the *sets* of constraints,
$(\forall X \in \chi_i)(X \leq P_i?)$ and $(\forall X \in \chi_{i+1})(P_i? \leq X)$. Note that no ordering con-
straints are imposed among any pair of time-points, X and Y, that belong to
the same set χ_i.

Consider the CSTN graph shown on the lefthand side of Fig. 3. One pos-
sible sequence of moves is: $(\{X\}, P?), (p = true), \{Y\}$, whose corresponding
ordering constraints are: $Z \leq X \leq P? \leq Y$. Another possible sequence
is: $(\emptyset, P?), (p = false), \{X, W\}$, whose corresponding ordering constraints are:
$Z \leq P? \leq \{X, W\}$.

The (abstract) game tree—or *simulation graph* [3]—is a branching tree with
finitely many nodes. It includes: (1) *agent nodes* (Agt-nodes) that represent
states where it is the agent's turn to "play" time-points; and (2) *environment
nodes* (Env-nodes) that represent states where it is the environment's turn to

---

[2] Similar assumptions have been made elsewhere [13,16]. In contrast, recent papers
address ϵ-dynamic controllability for CSTNs, where agent reaction times are bounded
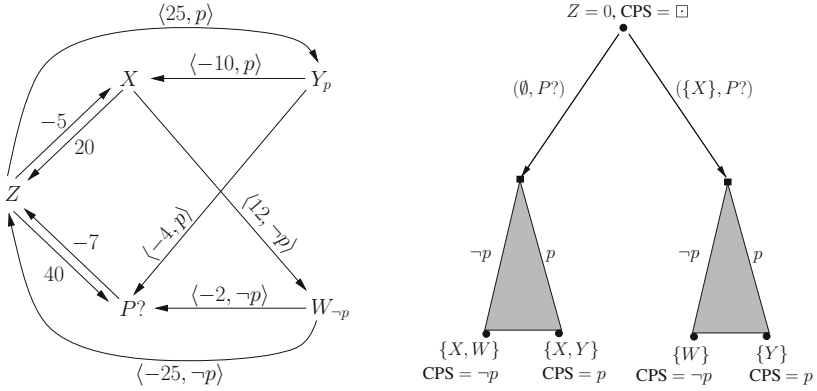below [7,13].

Fig. 3. A sample CSTN (left) and the corresponding game tree (right)

select a truth value for a propositional letter. The game tree for the sample CSTN is illustrated on the righthand side of Fig. 3. In the figure, each agent move is represented by an edge from an Agt-node ($\bullet$) to an Env-node ($\blacksquare$), and each *binary choice* for the environment is represented by a *hyper-edge* [15], shown as a shaded triangle, whose source is an Env-node, and whose *target set* is a pair of Agt-nodes: one for *true,* one for *false.*

Our algorithm's traversal of the game tree is inspired by Liu and Smolka's algorithm for finding minimal fixed points of dependency graphs [15]. Their algorithm has a forward phase that does depth-first search with backtracking; and a back-propagation phase that can interrupt the forward phase. The forward phase uses a global *last-in, first-out* queue of *forward edges* (i.e., moves); the processing of each forward edge typically involves pushing more forward edges onto the queue. The back-propagation phase (not to be confused with backtracking in the forward phase) is engaged by pushing a *backward edge* onto the queue. Processing backward edges involves propagating information back toward the root node, and continues as long as certain criteria are met. If the back-propagation peters out, the forward phase is automatically re-engaged by processing the topmost edge from the queue. Our algorithm similarly interleaves forward and backward phases but, as will be seen, uses multiple queues.

When it is the agent's turn, it can play any of the as-yet-unplayed time-points whose labels are entailed by the current partial scenario. For each move, $(\chi, P?)$, our algorithm effectively asks: "Is there an assignment for the time-points in $\chi \cup P?$ that can force a win *from this point onward?*" That question is not answered immediately. But notice that a win-forcing assignment for the time-points in $\chi \cup P?$ must be win-forcing regardless of the environment's subsequent choice for the value of p.

Although the question, "Is there a win-forcing assignment?", is not answered immediately, the constraints from the CSTN that apply to the time-points in $\chi \cup \{P?\}$, and any other already-played time-points, together with the ordering

constraints discussed above, do restrict the space within which such win-forcing assignments must reside—should they exist. This restricted space of possible win-forcing assignments can be represented by an STN, hereinafter called the *current STN.* A consistency check on the current STN can be used to prune moves that cannot be part of a winning strategy.

A leaf node in the game tree represents a state where all observation time-points whose labels are entailed by the CPS have already been played. At that point, the agent must play all of the as-yet-unplayed ordinary time-points whose labels are entailed by the CPS. Let χ be that (possibly empty) set of time-points. Now, if the current STN for that leaf node is consistent, the question "Is there an assignment for the time-points in χ that can force a win *from this point onward?*" is trivially "Yes", since any solution to the current STN will do. In other words, for a leaf node in the game tree, the solution set for the current STN represents not only the restricted domain within which any win-forcing assignment must reside, but in fact the actual set of win-forcing assignments for that node. Hereinafter, any set of win-forcing assignments shall be called a *win-set.*

The forward phase of our algorithm uses depth-first search with backtracking to find a sequence of moves that terminate in a leaf node whose current STN is consistent (i.e., whose win-set is non-empty). To illustrate the forward phase, recall the game tree from the righthand side of Fig. 3. Its root node has the following information:

- Z is the only time-point that has been "played";
- $\pi_0 = \boxdot$ is the current partial scenario;
- $\mathcal{T}_0 = \{Z, X, P?\}$, the time-points whose labels are entailed by $\pi_0 = \boxdot$; and
- $\mathcal{C}_0 = \{X \in [5, 20], P? \in [7, 40]\}$, the constraints whose labels are entailed by π_0.

From the root node, there are only two legal moves: $(\emptyset, P?)$ or $(\{X\}, P?)$. Let's explore the latter move. After this move, the current STN is $\mathcal{S}_1 = (\{Z, X, P?\}, \mathcal{C}_0 \cup \theta)$, where:

- $\theta = \{Z \leq X \leq P?\}$, the ordering constraints associated with the move $(\{X\}, P?)$.

The distance matrix $\mathcal{D}_1$ for $\mathcal{S}_1$ is shown in Fig. 4a. Any win-forcing assignment to time-points in $\{Z, X, P?\}$ must satisfy the constraints represented by this matrix. However, at this point, it is not known whether any win-forcing assignments exist.

Next, suppose that the environment chooses to set $p = true$. In that case, we get:

- $\{Z, X, P?\}$ are the time-points that have been played;
- $\pi_1^p = p$ is the current partial scenario; and
- $\mathcal{T}_1^p = \{Y\}$ is the set of unplayed time-points whose labels are entailed by $\pi_1^p = p$.
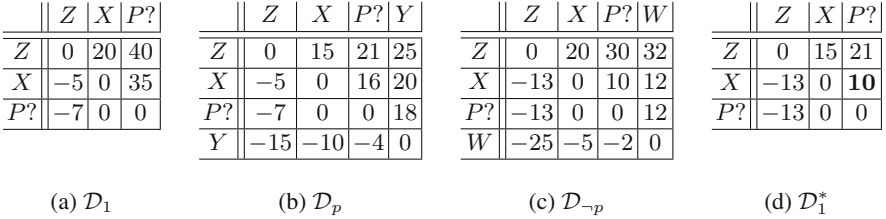
	Z	X	P?
Z	0	20	40
X	−5	0	35
P?	−7	0	0

(a) $\mathcal{D}_1$

	Z	X	P?	Y
Z	0	15	21	25
X	−5	0	16	20
P?	−7	0	0	18
Y	−15	−10	−4	0

(b) $\mathcal{D}_p$

	Z	X	P?	W
Z	0	20	30	32
X	−13	0	10	12
P?	−13	0	0	12
W	−25	−5	−2	0

(c) $\mathcal{D}_{\neg p}$

	Z	X	P?
Z	0	15	21
X	−13	0	**10**
P?	−13	0	0

(d) $\mathcal{D}_1^*$

Fig. 4. Distance matrices related to the running example

Since $\mathcal{T}_1^p$ contains no observation time-points, the agent's only option is to play the remaining time-point Y, ending the run with the current STN $(\mathcal{T}_1^p, \mathcal{C}_1^p \cup \theta_p)$, where:

- $\mathcal{C}_1^p = \mathcal{C}_0 \cup \{Y \leq 25, X - Y \leq -10, P? - Y \leq -4\}$; and
- $\theta_p = \theta \cup \{P? \leq Y\}$ are the relevant ordering constraints.

The distance matrix, $\mathcal{D}_p$, for this STN is shown in Fig. 4b. Because it is consistent, its solution set *is* the win-set for the leaf node at the bottom-right of the game tree in Fig. 3. However, "win-forcing" only means win-forcing *from that point onward.* Thus, whenever a leaf node with a non-empty win-set is found, the forward phase is interrupted by the back-propagation phase, whose purpose is to determine the constraints that must be imposed on preceding nodes to ensure that this non-empty win-set can be reached.

By Theorem 1, any solution to the *restriction* of $\mathcal{D}_p$ to $\{Z, X, P?\}$ can be extended to a solution to $\mathcal{D}_p$ over the time-points in $\{Z, X, P?, Y\}$. That is, any solution to the STN represented by the upper-lefthand 3×3 sub-matrix of $\mathcal{D}_p$, which involves only $\{Z, X, P?\}$, is necessarily win-forcing—as long as the environment subsequently chooses $p = true$. However, the agent must also force a win should the environment choose $p = false$. The relevant matrix for that case is $\mathcal{D}_{\neg p}$, shown in Fig. 4c. Its upper-lefthand 3×3 matrix must be satisfied by Z, X and $P?$ to ensure a win should the environment choose $p = false$. Since the agent wants to ensure a win no matter which value the environment chooses for p, a win-forcing assignment for Z, X and $P?$ must satisfy the constraints represented by *both* 3×3 sub-matrices. In other words, a win-forcing assignment must belong to the *intersection* of the solution sets represented by those sub-matrices. By Lemma 1, the distance matrix, $\mathcal{D}$, for that intersection can be computed as follows. First, the matrix $\mathcal{D}_1^*$, shown in Fig. 4d, is computed by taking the *element-wise minimum* of the corresponding sub-matrices and then running an all-pairs, shortest-paths computation on $\mathcal{D}_1^*$, which in this case only changes the 10 to an 8. The resulting matrix, $\mathcal{D} = \mathcal{D}_p \cap \mathcal{D}_{\neg p}$, is called the *back-propagation* of $\mathcal{D}_p$ and $\mathcal{D}_{\neg p}$.

For the CSTN from Fig. 3, the back-propagation of $\mathcal{D}_p$ and $\mathcal{D}_{\neg p}$ provides a non-empty win-set for the Env-node associated with the move $(\{X\}, P?)$. The distance matrix for that win-set can then be propagated back to the preceding

Agt-node—in this case, the root node—simply by restricting it to the time-points associated with that node—in this case, Z. Since the restriction of a consistent matrix is invariably consistent, propagating a win-set from an Env-node back to its parent Agt-node never fails. For the sample CSTN, the result is $[0]$, a consistent 1×1 matrix confirming that $Z = 0$ is a win-forcing assignment for the root node. Thus, the sample CSTN is DC. However, in many cases, back-propagation may not make it all the way back to the root node because, at some point, the intersection of the win-sets from two child Agt-nodes may be empty. In such cases, back-propagation stops and the forward phase is re-engaged.

Maintaining Lists of Win-Sets. Suppose that N is an Env-node whose two child Agt-nodes are N_p and $N_{\neg p}$, as illustrated in Fig. 5. Next, suppose that the forward search from N_p explores an agent move, m_1, that eventually leads to a leaf node with a non-empty win-set; and that the subsequent back-propagation of that win-set results in a non-empty win-set for N_p, represented by a matrix $\mathcal{D}_p^1$. If the forward search has yet to explore the Agt-node $N_{\neg p}$, then there can be no win-set for that node to intersect with $\mathcal{D}_p^1$. Thus, back-propagation of $\mathcal{D}_p$ must stop until forward search from $N_{\neg p}$ (e.g., by exploring a move M_1) leads to a leaf node with a non-empty win-set that can be subsequently back-propagated to a non-empty win-set $\mathcal{D}_{\neg p}^1$ for $N_{\neg p}$. Now, if $\mathcal{D}_p^1 \cap \mathcal{D}_{\neg p}^1$ is non-empty, back-propagation from N toward the root node can be re-started. However, if $\mathcal{D}_p^1 \cap \mathcal{D}_{\neg p}^1$ turns out to be empty, it need not imply that the Env-node N is a dead end, because there may be alternative agent moves that can be explored from N_p and $N_{\neg p}$. For example, some of the alternative moves from N_p (e.g., $m_2, m_3, \ldots$) may lead to additional win-sets for N_p, represented by matrices, $\mathcal{D}_p^2, \mathcal{D}_p^3, \ldots$. Similarly, various moves from $N_{\neg p}$ may lead to additional win-sets for N_p, represented by matrices, $\mathcal{D}_{\neg p}^2, \mathcal{D}_{\neg p}^3, \ldots$. Any pairing of some $\mathcal{D}_p^i$ with some $\mathcal{D}_{\neg p}^j$ for which the intersection $\mathcal{D}_p^i \cap \mathcal{D}_{\neg p}^j$ is non-empty will provide a non-empty win-set for the Env-node N, which may lead to successful back-propagation toward the root node. Since the algorithm cannot know in advance which of the win-sets for N_p and $N_{\neg p}$, if any, may contribute to non-empty win-sets for N, it maintains for each Agt-node a *list* of the win-sets that have been found so far for that node. Whenever a new win-set, $\mathcal{D}_p^i$, is found for N_p, it must be checked against each win-set, $\mathcal{D}_{\neg p}^j$ for $N_{\neg p}$, to determine whether the intersection, $\mathcal{D}_p^i \cap \mathcal{D}_{\neg p}^j$, is non-empty, whence back-propagation from the Env-node N could be re-started. The following properties of win-sets summarize the main points of the preceding discussions.
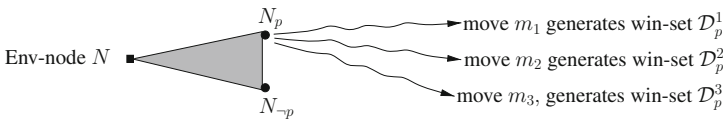

Fig. 5. Illustrating the need for lists of win-set matrices at Agt-nodes

(P1) Let N be an Agt-node; and $N_1, \ldots, N_k$ its child Env-nodes. If $\mathcal{D}_1, \ldots, \mathcal{D}_k$ are win-sets for those child nodes, and $\mathcal{T}_N$ is the set of time-points whose labels are entailed by the CPS at N, then for each i, $(\mathcal{D}_i|_{\mathcal{T}_N})$ is a win-set for N.

(P2) Let N be an Env-node; and N_p and $N_{\neg p}$ its child Agt-nodes. If $\mathcal{D}_p$ and $\mathcal{D}_{\neg p}$ are win-sets for N_p and $N_{\neg p}$, respectively, and $\mathcal{T}_N$ is the set of time-points whose labels are entailed by the CPS at N, then $(\mathcal{D}_p|_{\mathcal{T}_N}) \cap (\mathcal{D}_{\neg p}|_{\mathcal{T}_N})$ is a win-set for N.

(P1) follows from Theorem 1; (P2) follows from Lemma 1 and the fact that a win-forcing assignment for N must be win-forcing for *both* N_p and $N_{\neg p}$.

Incrementally Constructing Agent Moves. From any Agt-node, there may be exponentially many moves, each of the form $(\chi, P?)$, where χ is a possibly empty subset of OrdTPs, and $P?$ is an ObsTP. To increase efficiency, our algorithm *incrementally constructs* each agent move in a sequence of *steps.* For each step, a time-point is selected for incorporation into the nascent move. Each OrdTP that is selected is added to χ. When an ObsTP is (eventually) selected, the construction of the move $(\chi, P?)$ is completed. In this way, a move $(\chi, P?)$ from an Agt-node to an Env-node is represented by a sequence of *step-edges,* as illustrated in Fig. 6. The selection of an OrdTP is represented by a *forward OrdTP edge,* whose destination is an *intermediate node.* The selection of an ObsTP is represented by a a *forward ObsTP edge,* whose destination is an Env-node. In this way, each agent move is represented by a sequence of zero or more forward OrdTP edges, followed by a single ObsTP edge that terminates at an Env-node.
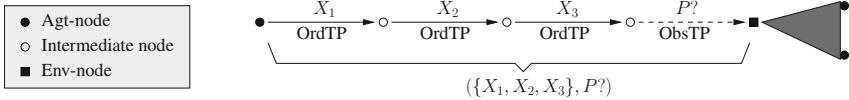
Fig. 6. A sequence of edges representing a transition from an Agt-node to an Env-node

Monte-Carlo Tree Search. At each step during the incremental construction of an agent move, the algorithm uses Monte-Carlo Tree Search (MCTS) [9] as a *variable-ordering heuristic.* One call to MCTS generates a list of step-edges to be pushed onto the Agt-node's queue. The step-edges are sorted so that the "best" step-edge will be popped first. If an OrdTP is selected, MCTS is run again to once again generate a sorted list of step-edges to be pushed onto the Agt-node's queue, and so on, until an ObsTp is selected, whence the construction of the move is completed, and the resulting Env-node can be created and explored. Further details of our use of MCTS are provided later on.

Limited Discrepancy Search. To avoid getting trapped in an unpromising portion of the search space, instead of doing ordinary depth-first search, the algorithm

uses *Limited Discrepancy Search* (LDS) [10], as follows. First, each call to MCTS generates an ordered list of step-moves. If the best step-move is explored, there is no penalty (or discrepancy). But if the algorithm backtracks and tries the next best move, a penalty of $+1$ is accumulated. Further backtracking, leading to exploring even worse moves, leads to higher penalties. Thus, the sequence of moves that is currently being explored, going all the way back to the root node, has an associated total path discrepancy. LDS with a limit of L ignores any move that would lead to a total path discrepancy greater than L. Since a winning strategy might not be obtained from LDS using a given limit L, the algorithm employs an *iterative deepening* version of discrepancy search where the limit L starts out at 0 (i.e., only best step-moves are explored at each step). If that search fails to find a winning strategy, the limit L is incremented and the algorithm tries again. This process continues until a winning strategy is found or the search space is exhausted.

Table 1. Pseudo-code for the `SG-DC-CHECK` algorithm for CSTNs (Part One)

```
SG-DC-CHECK:
   v₀ := root Agt-node;
   v₀.queue := MCTS(v₀), a sorted list of step-edges
   L := 0    (initial discrepancy limit);
   while(true)
      result := SEARCH(v₀, 0, L).
      if (result == non-empty win-set for v₀), then return DC;
      elseif (result == search_space_exhausted), then return non-DC;
      else L := L + 1.

SEARCH(v, d, L):   (v = Agt-node, d = accumulated discrepancy, L = current max discrepancy)
   new_winners := NIL
   while((new_winners == NIL) && (v.queue ≠ ∅))
      currEdge := pop(v.queue);        (currEdge is a forward or backward edge; see Table 2)
      when(d + discr(currEdge) ≤ L)
         new_winners := PROCESS-EDGE(v, currEdge, d).
   return new_winners.
```

Pseudo-code for the New DC-Checking Algorithm. The new DC-checking algorithm for CSTNs is called `SG-DC-CHECK` (for "simulation-graph DC-checking"). Pseudo-code for the `SG-DC-CHECK` algorithm is given in Tables 1 and 2. The `SG-DC-CHECK` algorithm calls the `SEARCH` method on the root Agt-node v_0, whose sorted list of initial step-moves has been generated by MCTS and pushed onto v_0's queue. The `SEARCH` method processes (forward or backward) edges from the queue of as-yet-unprocessed edges. The behavior of the `PROCESS-EDGE` method depends on the type of edge (forward or backward; OrdTP, ObsTP or Env) to be processed. Processing a forward OrdTP edge creates a new intermediate node; and uses MCTS to generate a sorted list of legal step-moves to be pushed onto the Agt-node's queue. Processing a forward ObsTP edge creates a new Env-node and

Table 2. Pseudo-code for the `SG-DC-CHECK` algorithm for CSTNs (Part Two)

LEGEND:
> v is an Agt-node, v_c is an Agt-node or intermediate node;
> v_E is an environment node, X is an OrdTP, $P?$ is an ObsTP;
> d is the accumulated discrepancy, $\hat{v}$ is an Agt-node;
> *newWinner* is a win-set matrix, and *newWinners* is a list of win-set matrices.

PROCESS−EDGE($v, fwdOrdTP(v_c, X), d$):
  Create intermediate node with X played; and accumulate (partial) ordering constraints;
  `when`(constraints consistent)
    `push`(MCTS(v_c), $v.queue$).   (i.e., push sorted list of legal step-moves onto v's queue)

PROCESS−EDGE($v, fwdObsTP(v_c, P?), d$):
  Create Env-node v_E, and compute distance matrix $\mathcal{D}_E$ for its current STN;
  `when`($\mathcal{D}_E$ is consistent)
    `push`($envHyperEdge(v_E, P?), v.queue$).

PROCESS−EDGE($v, envHyperEdge(v_E, P?), d$):
  `if`(first time visiting this hyper-edge):
    Create child Agt-nodes, v_p and $v_{\neg p}$, and corresponding distance matrices, $\mathcal{D}_p$ and $\mathcal{D}_{\neg p}$;
    `when`(element-wise min. $\mathcal{D}'_E$ of restricted matrices $\mathcal{D}'_p$ and $\mathcal{D}'_{\neg p}$ is consistent):
      `for each` $\hat{v} \in \{v_p, v_{\neg p}\}$,
        `if`($\hat{v}$ is a leaf node), $\hat{v}.finished := true$, and $\hat{v}.winners := \{\mathcal{D}'_E\}$;
        `else`, $\hat{v}.queue := $MCTS($\hat{v}$);
      `if` ($v_p.finished$ && $v_{\neg p}.finished$), `push`($bkwdObsEdge(v_E, \mathcal{D}'_E), v.queue$);
      `else push`($envHyperEdge(v_E, P?), v.queue$);
  `else` (i.e., re-visiting this edge):
    `if`($v_p.finished$ && $v_{\neg p}.finished$), then `return` NIL;
    `else select` $\hat{v} \in \{v_p, v_{\neg p}\}$ such that $\hat{v}.finished = $ NIL;
      `when` $winner :=$ SEARCH($\hat{v}$) $\neq \emptyset$,
        `push`($bkwdEnvEdge(v_E, \hat{v}, winner), v.parent.queue$).

PROCESS−EDGE(v,bkwdEnvEdge($v_E, \hat{v}, newWinner), d$):   ($d$ ignored during back-prop)
  $\tilde{v} :=$ the *other* Agt-node child of v_E;   ($\tilde{v}$ and $\hat{v}$ are the children of v_E)
  $winAcc := \emptyset$;   ($winAcc$ will accumulate winners for v_E)
  `for each` $oldWinner \in \tilde{v}.winners$,
    $possWinner := newWinner \cap oldWinner$;
    `if` $possWinner \nsubseteq \bigcup_{W \in v_E.winners} W$, `push`($possWinner, accWinners$);
  `push` ($envHyperEdge(v_E, P?), v$);   (if back-prop sputters, must revisit forward hyper-edge)
  `when`($winAcc \neq \emptyset$), `push`($bkwdObsEdge(v_E, winAcc), v.queue$);   (continue back-prop)
                                 (else forward phase automatically continues)

PROCESS−EDGE($v, bkwdObsEdge(v_E, newWinners), d$):   ($d$ ignored during back-prop)
  $winAcc := \emptyset$;   ($winAcc$ will accumulate winners for v)
  `for each` $newWinner \in newWinners$,
    $restrictedWinner := newWinner|_{v.\mathcal{T}}$ (i.e., restrict winner to time-points in v)
    `if` $restrictedWinner \nsubseteq \bigcup_{W \in v.winners} W$, `push`($restrictedWinner, winAcc$);
  `if`(($winAcc \neq \emptyset$) && ($rootNode?(v)$)) `return` DC;   (Non-empty win-set for root node!)
  `else if`($winAcc \neq \emptyset$) `push`($bkwdEnvEdge(v_E, winAcc), v.queue$).   (continue back-prop)

pushes an *envHyperEdge* (environment hyper-edge) onto the Agt-node's queue. The first time an envHyperEdge is processed, two Agt-node children are created. If both child nodes happen to be leaf nodes, then their restricted win-sets are intersected and, if non-empty, form a win-set for the parent Env-node. In that case, a *bkwdObsEdge* (i.e., backward observation edge) is pushed onto the queue of the Env-node's parent Agt-node to initiate back-propagation of that new win-set. (Env-nodes do not have queues.) Otherwise, the envHyperEdge that is currently being processed is pushed back onto the queue to ensure that it is immediately re-visited. Whenever an envHyperEdge is re-visited, it calls the SEARCH method on one of the child Agt-nodes to see whether a new win-set can be generated. If so, then a *bkwdEnvEdge* (i.e., backward environment edge) is pushed onto the parent Agt-node's queue to back-propagate that new win-set. If the back-propagation peters out (i.e., fails to reach the root node of the entire simulation graph), then the envHyperEdge will be re-visited to see whether additional win-sets for the child Agt-nodes might be found, as discussed with Fig. 5.

As already indicated, back-propagation is implemented by processing *bkwdEnvEdges* and *bkwdObsEdges*. Processing a bkwdEnvEdge for a given Env-node v_E takes a new win-set for one of the child Agt-nodes $\hat{v}$ and intersects it with the existing win-sets for the other child Agt-node $\tilde{v}$. If any new win-sets are generated that are not subsumed by any of the existing win-sets for v_E, then those win-sets are packaged into a bkwdObsEdge that is pushed onto the parent Agt-node's queue to ensure that back-propagation continues. Processing a bkwdObsEdge takes new win-sets for an Env-node v_E and restricts the corresponding distance matrices to the time-points relevant to the parent Agt-node. If new win-sets for the parent Agt-node are generated, then a bkwdEnvEdge is pushed onto the queue of the grandparent Agt-node, and back-propagation continues. If the back-propagation ever reaches the root Agt-node with a non-empty win-set, then the network is declared to be DC. If the search fails to find a win-set for the root node, then the discrepancy limit L is increased. That process continues until a win-set for the root node is found, or the search space is exhausted.

When run without using MCTS as a node-ordering heuristic, and without using iterative-deepening limited discrepancy search, the SG-DC-CHECK algorithm performs quite poorly—even when using consistency checks on the current STN for each node to prune bad moves. The reason is clear: there are exponentially many paths through the search tree, most of which typically do not lead to a leaf node with a non-empty win-set. Therefore, our algorithm uses MCTS to pick the "best" time-points to insert into agent moves before they are played, and uses iterative-deepening discrepancy search to avoid wasting time in poor areas of the search tree. Our implementation of MCTS is closely based on Gelly and Silver's two-player UCT algorithm, combined with their *all moves as first* (AMAF) technique for *rapid action value estimation* (RAVE) [9]. In the basic UCT algorithm, moves for the player and environment are explored, with a balance between exploration and exploitation. Each node in the incrementally

expanding tree of explored moves keeps track of how many times that node was visited, and the cumulative award that node received from those visits. When a previously unexplored node is reached, a random play-out is performed until either (1) a leaf node with a non-empty win-set is found; or (2) a node whose current STN is inconsistent is reached. In the first case, the win is worth one point, and the algorithm's *backup* procedure increments the number of wins and the number of visits for the nodes seen during that run. In addition, the AMAF-RAVE technique treats each ordinary time-point from the first agent move $(\chi, P?)$ as a *first* move, which is justified because those time-points are not ordered. In the second case, the value of the dead-end node is proportional to the depth reached.

5   Empirical Evaluation

The fastest DC-checking algorithm for CSTNs from the literature is from our earlier work [13]. For convenience, that algorithm shall be called the DC-CHECK algorithm. It is based on a more traditional style of constraint propagation, extended to accommodate labeled constraints. This section compares the performance of the new SG-DC-CHECK algorithm and the existing DC-CHECK algorithm.

First, it must be acknowledged that the DC-CHECK algorithm performs very well on networks that are either (1) very loosely constrained or (2) inconsistent. In the first case, there is not much constraint propagation to perform, so the algorithm runs quickly. In the second case, negative loops signalling non-DC can often be found very quickly. In contrast, the SG-DC-CHECK algorithm typically explores an exponential number of nodes in the simulation graph even for loosely constrained networks, because it must ensure the existence of a win-forcing strategy; and for non-DC networks, it must *exhaust* the search space to prove that no winning strategy can exist. Therefore, this section focuses primarily on moderately constrained networks.

The generation of CSTN instances was based on random workflow schema generated by the ATAPIS toolset [14]. It is hoped that instances generated in this way may be closer to examples that might be encountered in the real world. Thirty consistent CSTNs were randomly generated. Each CSTN had between 107 and 155 time-points, between 8 and 16 observation time-points, and between 428 and 970 constraints. Both DC-checking algorithms were implemented in Lisp and run on Intel Core i7-4790 machines with 3.6 GHz processors (4 cores/8 threads), running Ubuntu 14.04.04 LTS (kernel version 3.16.0-67-generic) and Allegro Common Lisp (ACL) Enterprise Edition, version 8.1. To dramatically reduce the number of redundant labeled edges stored during constraint propagation, the DC-CHECK algorithm was improved to store labeled edges in *subsumption hierarchies,* one hierarchy for each pair of time-points. Because the DC-CHECK algorithm is deterministic, it was run only once on each network. In contrast, the SG-DC-CHECK algorithm employs randomness, both from the use of MCTS to determine "best" moves, and in deciding which of the two child

nodes to explore when processing an envHyperEdge. Therefore, SG-DC-CHECK was run five times on each network.

The average run-time of the SG-DC-CHECK algorithm was better than that of the DC-CHECK algorithm for 11 of the 30 instances, while the DC-CHECK algorithm was faster on 18 of the 30 instances. For one instance, *both* algorithms timed out (over 10 min). Of the 18 instances where the DC-CHECK algorithm was faster, the SG-DC-CHECK algorithm timed out at least once on 6 instances. Over the 23 instances where neither algorithm timed out, the mean run times were 40.0 s for DC-CHECK and 53.8 s for SG-DC-CHECK, with standard deviations of 80.8 and 89.6, respectively. Furthermore, over the 29 instances where the DC-CHECK algorithm did not time out, the mean run time of DC-CHECK was 38.4 s, with a standard deviation of 80.7, but over the same 29 instances, the *minimum* run time of the two algorithms (actual for DC-CHECK, average for SG-DC-CHECK over 5 trials) had a mean of 22.1 s with a standard deviation of 24.0. This suggests that running the two algorithms in parallel, and taking the answer obtained by whichever algorithm finishes sooner, has the potential to dramatically improve the timing performance.

Finally, to demonstrate a shortcoming of the DC-CHECK algorithm, recall the small CSTN from Fig. 2. The loops between X and P?, and Y and Q?, are examples of what are called *negative q-loops* [13]. They do *not* necessarily cause a network to be non-DC. In fact, the CSTN in Fig. 2 *is* DC. However, they can lead DC-CHECK to do an incredible amount of constraint propagation. Each loop reinforces the other, gradually increasing the lower bounds on P? and Q? until they eventually approach the value of the lower bound on W?, which is 100. (To understand why, see the constraint-propagation rules in our earlier work [13].) If the lower bound on W? is increased to 1000, then DC-CHECK takes almost three minutes to determine that the network is DC. In contrast, the SG-DC-CHECK algorithm solves this network in 80 ms, regardless of the size of the lower bound on W? This example highlights the fact that the two algorithms have very different strengths and weaknesses. It suggests that further study of both algorithms is warranted. Both algorithms may achieve superior results in the future by improving their implementations. For example, improving the consistency checking and tuning the parameters governing the Monte-Carlo Tree Search may dramatically improve the performance of the SG-DC-CHECK algorithm.

6 Conclusion

The SG-DC-CHECK algorithm continues a line of research dating back to the literature on finding minimal fixed-points for dependency graphs [15], synthesizing controllers for TGAs [3], and DC checking for DTNUs [6]. Like the DC-checking algorithm for DTNUs, the SG-DC-CHECK algorithm: (1) searches through an abstract *simulation graph;* (2) manages its traversal through that graph by keeping track of edges waiting to be processed, computing winning sets for nodes, and keeping track of dependencies among nodes; (3) keeps track of ordering

constraints and performs consistency checks for pruning bad moves; and (4) in successful instances, can be used to generate a dynamic execution strategy. However, adapting the high-level approach to CSTNs required the development of significant novel representations and techniques. For example: (1) TGAs are not used; (2) winning sets are represented by STNs, not TGA clock zones; (3) consistency checking is done using STN algorithms; (4) the simulation graph includes hyper-edges that represent the *true* and *false* branches for observations; (5) STN restriction and intersection are used to back-propagate win-sets; and (6) Monte-Carlo Tree Search and Limited Discrepancy Search are used to guide the search.

References

1. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, pp. 825–885. IOS Press, Amsterdam (2009)
2. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
3. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005)
4. Cimatti, A., Hunsberger, L., Micheli, A., Posenato, R., Roveri, M.: Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In: TIME 2014. pp. 27–36. IEEE Computer Society, September 2014
5. Cimatti, A., Hunsberger, L., Micheli, A., Roveri, M.: Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014) (2014)
6. Cimatti, A., Micheli, A., Roveri, M.: Dynamic controllability of disjunctive temporal networks: validation and synthesis of executable strategies. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016) (2016)
7. Comin, C., Rizzi, R.: Dynamic consistency of conditional simple temporal networks via mean payoff games: a singly-exponential time DC-checking. In: 22st International Symposium on Temporal Representation and Reasoning (TIME 2015), pp. 19–28. IEEE CPS, September 2015
8. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. **49**(1–3), 61–95 (1991). http://dx.doi.org/10.1016/0004-3702(91)90006-6
9. Gelly, S., Silver, D.: Monte-carlo tree search and rapid action value estimation in computer go. Artif. Intell. **175**, 1856–1875 (2011)
10. Harvey, W., Ginsberg, M.: Limited discrepancy search. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 607–613 (1995)
11. Hunsberger, L.: Group decision making and temporal reasoning. Ph.D. thesis, Harvard University, available as Harvard Technical Report TR-05-02 (2002)
12. Hunsberger, L., Posenato, R., Combi, C.: The dynamic controllability of conditional STNs with uncertainty. In: Proceedings of the Workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx), ICAPS-2012, pp. 1–8 (2012)

13. Hunsberger, L., Posenato, R., Combi, C.: A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In: 22st International Symposium on Temporal Representation and Reasoning (TIME 2015), pp. 4–18. IEEE CPS, September 2015

14. Lanz, A., Reichert, M.: Enabling time-aware process support with the atapis toolset. In: Limonad, L., Weber, B. (eds.) Proceedings of the BPM Demo Sessions 2014. CEUR Workshop Proceedings, vol. 1295, pp. 41–45. CEUR (2014)

15. Liu, X., Smolka, S.A.: Simple linear-time algorithms for minimal fixed points. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 53–66. Springer, Heidelberg (1998)

16. Morris, P.: Dynamic controllability and dispatchability relationships. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 464–479. Springer, Heidelberg (2014)

17. Morris, P.H., Muscettola, N., Vidal, T.: Dynamic control of plans with temporal uncertainty. In: Nebel, B. (ed.) Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), pp. 494–502. Kaufmann (2001)

18. Tsamardinos, I., Vidal, T., Pollack, M.E.: CTP: a new constraint-based formalism for conditional, temporal planning. Constraints **8**, 365–388 (2003)

On Finding Minimum Satisfying Assignments

Alexey Ignatiev[1,2]([✉]), Alessandro Previti[1], and Joao Marques-Silva[1]

[1] LaSIGE, Faculty of Science, University of Lisbon, Lisbon, Portugal
{aignatiev,jpms}@ciencias.ulisboa.pt, apreviti.research@gmail.com
[2] ISDCT SB RAS, Irkutsk, Russia

Abstract. Given a Satisfiability Modulo Theories (SMT) formula, a minimum satisfying assignment (MSA) is a partial assignment of minimum size that ensures the formula is satisfied. Minimum satisfying assignments find a number of practical applications that include software and hardware verification, among others. Recent work proposes the use of branch-and-bound search for computing MSAs. This paper proposes a novel counterexample-guided implicit hitting set approach for computing one MSA. Experimental results show significant performance gains over existing approaches.

1 Introduction

For a propositional formula, represented in conjunctive normal form (CNF), a minimum size prime implicant can be computed with a logarithmic number of calls to a SAT oracle [18], e.g. by solving unweighted partial maximum satisfiability. For arbitrary propositional formulae it is unclear how to solve the problem. Nevertheless, for formulas expressed in decidable fragments of first order logic (FOL), the same problem has been investigated as computing a *minimum satisfying assignment* (MSA), which represents a partial assignment of minimum cost to the formula's variables such that, for any assignment to the remaining variables, the formula is true [8]. The MSA problem finds application in software verification, hardware verification, but also in abductive inference [7]. Recent work identified other uses of MSAs [15,24].

Earlier work proposed a branch-and-bound algorithm for computing MSAs of logic formulas expressed in decidable fragments of FOL, being applied in relatively small-scale test cases. Given the potential for wider use in software verification, but also in abductive inference, and with the goal of targeting more challenging problem instances, this paper investigates alternative approaches for computing MSAs. Concretely, the paper builds on the recent work on implicit hitting sets [5,12–14,16,19,23], and develops an implicit hitting set approach for computing MSAs, where implicit hitting sets are obtained by refining identified counterexamples to the general goal of computing one MSA. The experimental results, obtained on existing but also novel suites of problem instances, show performance gains when compared with the existing branch-and-bound approach.

The paper is organized as follows. Section 2 introduces the notation used throughout the paper. The use of implicit hitting sets for computing MSAs is

detailed in Sect. 3, resulting in a new tool referred to as MINT. Section 4 compares the MINT solver with the existing tool MISTRAL [8]. Section 5 concludes the paper and identifies research directions.

2   Preliminaries

This section introduces the concepts used throughout the paper. Standard definitions commonly used in first-order logic apply. We follow [3,8], since we build on this earlier work. The paper assumes basic knowledge of first-order logic, but some basic definitions are presented to make the paper more self-contained. A first-order theory $\mathcal{T}$ is a set of first-order sentences over a signature $\mathcal{S}$, where the signature $\mathcal{S}$ specifies a set of predicates and function constants. A first-order model $\mathcal{M}$ is a pair $\langle \mathcal{U}, \mathcal{I} \rangle$, where the set $\mathcal{U}$ represents a *universe*, and $\mathcal{I}$ represents an *interpretation* that assigns a semantics to every symbol in $\mathcal{S}$. $\mathcal{V}$ denotes the set of variables (which are distinct from $\mathcal{S}$). Given a model $\mathcal{M}$, a *valuation* ω is a partial map from $\mathcal{V}$ to $\mathcal{U}$. For simplicity, we assume $\mathcal{V}$ to be the set of variables that occur free in the formula. In what follows, $\mathcal{F}$ denotes a first-order formula modulo a theory defined over variables $\mathsf{var}(\mathcal{F})$. If ω is a valuation in $\mathcal{V} \to \mathcal{U}$, we write $\mathcal{M}, \omega \vDash \mathcal{F}$ to indicate that the formula $\mathcal{F}$ is true, according to the usual semantic of first-order logic, in model $\mathcal{M}$, with ω giving the valuation of the free variables in $\mathcal{F}$. We say that $\mathcal{M}$ is a model of $\mathcal{F}$ when every sentence of $\mathcal{F}$ is true in $\mathcal{M}$.

Definition 1. *Formula $\mathcal{F}$ is satisfiable modulo $\mathcal{T}$ when there exists a model $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$ of $\mathcal{T}$ and an assignment $\omega \in \mathcal{V} \to \mathcal{U}$ such that $\mathcal{M}, \omega \vDash \mathcal{F}$. We say that the pair $\langle \mathcal{M}, \omega \rangle$ is a* satisfying assignment (SA) *for $\mathcal{F}$.*

A concept used throughout the paper is that of *partial satisfying assignment*:

Definition 2. *A* partial satisfying assignment *for a formula $\mathcal{F}$ is a pair $\langle \mathcal{M}, \omega \rangle$, where $\mathcal{M}$ is a model and ω is a valuation over $\mathcal{M}$ such that $dom(\omega) \subseteq \mathcal{V}$ and such that for any valuation of $\alpha \in \mathcal{V} \setminus dom(\omega) \to \mathcal{U}$, we have that $\langle \mathcal{M}, (\omega \cup \alpha) \rangle$ is a satisfying assignment for $\mathcal{F}$.*

Definition 3. *A partial satisfying assignment $\langle \mathcal{M}, \omega \rangle$ for $\mathcal{F}$ is said to be minimal (mSA) if for any valuation $\alpha \in \mathcal{V} \to \mathcal{U}$ s.t. $dom(\alpha) \subset dom(\omega)$, the pair $\langle \mathcal{M}, \alpha \rangle$ is not a satisfying assignment for $\mathcal{F}$. A Minimum Satisfying Assignment (MSA) is an mSA of smallest size.*

Note that an MSA can be defined more generally in terms of a cost function. In this paper we will refer to MSAs in terms of their size (i.e. the cost associated with each variable is 1). We conclude this section with the definition of a hitting set:

Definition 4. *Given a collection Γ of sets from a universe $\mathbb{U}$, a hitting set h for Γ is a set such that $\forall S \in \Gamma, h \cap S \neq \emptyset$.*

A hitting set is *minimal* when none of its subsets is a hitting set. Let us denote with $\mathrm{HS}(\Gamma)$ the set of all hitting sets of Γ. Then a hitting set $h \in HS(\Gamma)$ is said to be a *minimum* hitting set if $\forall h' \in HS(\Gamma)$ we have that $|h| \leq |h'|$.

3   Computing One MSA with Implicit Hitting Sets

This section proposes an algorithm for computing one MSA by exploiting an implicit hitting set approach, following the ideas of [21] and similar in spirit to related recent work in different areas [5,12–14,16,19,23]. However, in contrast to earlier work related with propositional formulas, our approach exploits implicit hitting sets for selecting sets of variables and not sets of clauses.

The goal of computing one MSA is to find a minimum size set $X \in \mathsf{var}(\mathcal{F})$ such that $\exists_X \forall_Y . \mathcal{F}$ holds, with $Y = \mathsf{var}(\mathcal{F}) \setminus X$. Let us consider the set of sets $\mathcal{J}$, where each set $I \in \mathcal{J}$ represents the complement of one counterexample to our goal, i.e. each set $I \in \mathcal{J}$ is the complement of X_{cex} such that $\exists_{X_{\mathrm{cex}}} \forall_Y . \mathcal{F}$ is false. If a solution is to exist, it must use at least one variable from each set I; otherwise, we would be repeating the complement of set I, namely X_{cex}, and we know there is no solution in that case. Thus, any solution set $X \in \mathsf{var}(\mathcal{F})$ for which $\exists_X \forall_Y . \mathcal{F}$ is true, must *hit* any set $I \in \mathcal{J}$; otherwise, set X would not be a solution. Moreover, we can make each set to hit stronger, by finding a *maximal* counterexample, i.e. by *growing* X_{cex} [14] (and, hence, *reducing* its complement $I \in \mathcal{J}$). Thus, the minimal hitting set duality relation between goal sets X and reduced complements $I \in \mathcal{J}$ of counterexamples X_{cex} can be devised, following the ideas of [21]. In practice, it is unrealistic in many cases to explicitly represent the set of sets $\mathcal{J}$. Thus, the sets to hit are generated on demand, and this explains why the approach is referred to as an *implicit hitting set* approach. The proposed algorithm is depicted in Algorithm 1. The remainder of this section formalizes the approach outlined above.

Definition 5. *Given a formula $\mathcal{F}$ modulo theory $\mathcal{T}$ s.t. formula $\mathcal{F}$ is defined over set of variables $\mathsf{var}(\mathcal{F}) = X \cup Y$ and $\exists_X \forall_Y . \mathcal{F}$ is true, set Y is called a* universal subset (US) *for formula $\mathcal{F}$ and set X is called an* existential subset (ES) *for $\mathcal{F}$.*

Algorithm 1. MSA algorithm

    input : Formula $\mathcal{F}$
    output: One MSA of $\mathcal{F}$

1 $\mathcal{H} \leftarrow \emptyset$
2 **while** true **do**
3 $X \leftarrow \mathtt{MinHS}(\mathcal{H})$
4 $Y \leftarrow \mathsf{var}(\mathcal{F}) \setminus X$
5 $(\mathsf{st}, \mu_X) \leftarrow \mathtt{Solve}(\exists_X \forall_Y . \mathcal{F})$
6   **if** st **then**
7   **break**
8   **else**
9 $I \leftarrow \mathsf{var}(\mathcal{F}) \setminus \mathtt{Grow}(X, \mathcal{F})$
10 $\mathcal{H} \leftarrow \mathcal{H} \cup I$
11 **return** MSA $\leftarrow \mu_X$

An existential subset X (universal subset Y, resp.) is said to be minimal ES or MinES (maximal US or maxUS, resp.) if $\exists_{X \setminus \{z\}} \forall_{Y \cup \{z\}} . \mathcal{F}$ is false for any $z \in X$. An ES $\mathcal{E}$ of minimum size (US $\mathcal{U}$ of maximum size, resp.) is called a minimum existential subset or MinES (maximum universal subset or MaxUS, resp.). Observe that given an existential subset for formula $\mathcal{F}$ of the smallest size (i.e. a MinES), one can easily extract an MSA for $\mathcal{F}$. Indeed, it assigns variables of the existential subset satisfying $\mathcal{F}$, which can be done by calling a decision procedure for $\mathcal{F}$, i.e. an SMT oracle.

Definition 6. *A falsifying subset (FS) for a formula $\mathcal{F}$ modulo theory $\mathcal{T}$ is a set of variables $Y \subseteq \mathsf{var}(\mathcal{F})$ such that $\exists_X \forall_Y . \mathcal{F}$ is false.*

As usually, we identify with minFS and MinFS the minimal and minimum falsifying set, respectively. We can now introduce an important proposition highlighting the existing duality between the minESes and the minFSes of a formula:

Proposition 1. *Given a formula $\mathcal{F}$, let $minES(\mathcal{F})$ and $minFS(\mathcal{F})$ be the set of all minESes and minFSes of F. Then the following holds:*

1. *A subset $X \subseteq \mathsf{var}(\mathcal{F})$ is a minES for $\mathcal{F}$ iff X is a minimal hitting set of $minFS(\mathcal{F})$.*
2. *A subset $Y \subseteq \mathsf{var}(\mathcal{F})$ is a minFS for $\mathcal{F}$ iff Y is a minimal hitting set of $minES(\mathcal{F})$.*

The intuition[1] for the first statement is the following (a dual argument can be used for the second one). We know that since X is a minES for $\mathcal{F}$ then $\exists_X \forall_Y . \mathcal{F}$ is true. This means that for any minFS $Y' \in minFS(\mathcal{F})$ we have that $Y' \not\subseteq Y$, which implies that for any $Y' \in minFS(\mathcal{F})$ at least one variable of Y' is in X. If X is a minimal hitting set of $minFS(\mathcal{F})$ then at least one variable of every $Y' \in minFS(\mathcal{F})$ is in X. This means that $\exists_X \forall_Y . \mathcal{F}$ is true since $Y' \not\subseteq Y$ for any $Y' \in minFS(\mathcal{F})$.

Proposition 2. *A subset $X \subseteq \mathsf{var}(\mathcal{F})$ is a MinES for $\mathcal{F}$ iff X is a minimum hitting set of $minFS(\mathcal{F})$.*

Proposition 2 enables us to compute an MSA once we have the set $minFS(\mathcal{F})$. However, computing $minFS(\mathcal{F})$ is not always feasible due to the size of the set. Thus, the idea is to compute a subset of $minFS(\mathcal{F})$, from which an MSA can be extracted. However, a minimum hitting set X on a subset of $S \subseteq minFS(\mathcal{F})$ does not necessarily correspond to a MinES for $\mathcal{F}$. For X to be a MinES for $\mathcal{F}$ we need two conditions:

Proposition 3. *Let $Y = \mathsf{var}(\mathcal{F}) \setminus X$. If (1) X is a minimum hitting set of $S \subseteq minFS(\mathcal{F})$ and (2) $\exists_X \forall_Y . \mathcal{F}$ is true, then X is a MinES of $\mathcal{F}$.*

---

[1] Proofs of Propositions 1 and 2 are omitted here due to lack of space. Note that they can be constructed following the ideas of [12,22] where similar proofs are presented.

Proof. Since $\exists_X \forall_Y . \mathcal{F}$ is true we have that, by definition, X is an ES. We know that an ES is an hitting set of $minFS(\mathcal{F})$ (see Proposition 1). Since X is a *minimum* hitting set of $S \subseteq minFS(\mathcal{F})$ we have that X is also a minimum hitting set of $minFS(\mathcal{F})$. By Proposition 2, it follows that X is a MinES for $\mathcal{F}$.   □

Condition (2) of Proposition 3 is checked at line 5 of Algorithm 1 when the SMT oracle is invoked. If this oracle call returns true (line 6) then X is a MinES for $\mathcal{F}$ and, thus, an MSA can be extracted. Otherwise, X is extended[2] (line 9) and its complement (minFS) is added to the set $\mathcal{H}$, which contains the set of minFSes computed so far.

Table 1. MSA computation for $\mathcal{F} = ((a+b \geq 0) \vee (c \leq 0)) \wedge ((a+b \geq 0) \vee (b-a \leq 0))$

MinHS($\mathcal{H}$)	Solve($\exists_X \forall_Y . \mathcal{F}$)	$I \leftarrow$ var($\mathcal{F}$) $\setminus$ Grow($X, \mathcal{F}$)	$\mathcal{H} = \mathcal{H} \cup I$
$X \leftarrow \{\emptyset\}$	false	$I \leftarrow \{b, c\}$	$\{\{b, c\}\}$
$X \leftarrow \{b\}$	false	$I \leftarrow \{a\}$	$\{\{b, c\}, \{a\}\}$
$X \leftarrow \{a, c\}$	true	$\{a = 1, c = 0\}$ is an MSA of $\mathcal{F}$	

An example of a run of Algorithm 1 is shown in Table 1. Column 1 contains the set X of variables identified by the minimum hitting set of $\mathcal{H}$. Whenever Solve($\exists_X \forall_Y . \mathcal{F}$) returns true, the set X is extended by the Grow procedure (column 3) and its complement (minFS) is assigned to I. The last column shows the current $\mathcal{H}$, representing the set of all minFSes computed so far. In the last row, Solve($\exists_{\{a,c\}} \forall_{\{b\}} . \mathcal{F}$) returns true, with $a = 1, c = 0$. This means that $\{a, c\}$ is a MinES, and $a = 1, c = 0$ is an MSA.

4   Preliminary Experimental Results

This section evaluates the proposed approach to computing a minimum satisfying assignment. The experiments were performed in Ubuntu Linux on an Intel Xeon E5-2630 2.60 GHz processor with 64 GByte of memory. The time limit was set to 3600 s and the memory limit to 10 GByte.

The proposed approach was implemented in a prototype called MINT (**Mini**-mum satisfy**IN**g assignmen**T** extractor). The MINT MSA extractor is written as a Python script, which instruments the interaction between a minimum hitting set enumerator and an SMT solver, as described in Algorithm 1. The minimum hitting set enumerator (see line 3 of Algorithm 1) was implemented as an *incremental* MaxSAT solver[3] following the ideas of [17]. The MaxSAT solver was

---

[2] The Grow procedure is implemented as a sequence of SMT oracle calls, each increasing set X.

[3] Indeed, one can observe that Algorithm 1 requires the minimum hitting set solver to report new hitting sets on demand, i.e. when a new counterexample is detected. This can be done in an incremental fashion [10], e.g. by adding new clauses when necessary and computing new solutions on demand while keeping all the information found during the previous calls.

implemented on top of the well-known SAT solver Glucose 3.3 [1]. The MINT
extractor was implemented in the PySMT framework [11], which enables it to
use any SMT solver capable of dealing with the theories of input SMT formulas,
e.g. Z3 [6], CVC4 [2], Yices2 [9], etc. In contrast, the state-of-the-art MSA solver
MISTRAL [8] can work only with formulas in the theory of linear arithmetic
over integers (LIA formulas). In the experimental evaluation, CVC4 was used in
MINT as a backend SMT solver because CVC4[4] performed reasonably well in
the SMT competition SMT-COMP15[5] winning a few benchmark subcategories
in the QF_LIA category, and it also supports LIA formulas with quantifiers.

Additionally, an improved version of MINT was implemented, which is
referred to as MINT+. The only difference between MINT and MINT+ is that
MINT+ tries to bootstrap the main algorithm with sets of size 1 that need
to be hit in order to get an MSA. More precisely, the procedure traverses all
variables of $y \in \mathsf{var}(\mathcal{F})$ and decides satisfiability of the formula $\forall y. \mathcal{F}$. If the
formula is unsatisfiable than set $\{y\}$ is a MinFS of $\mathcal{F}$ of size 1, i.e. each MSA
of $\mathcal{F}$ necessarily hits it. The bootstrapping procedure is called before running
Algorithm 1. To assess the efficiency of MINT and MINT+, they were compared
to the state-of-the-art MSA extractor MISTRAL.[6]

4.1   Original Benchmark Instances

To evaluate the performance of MINT and MINT+, we used the benchmark set
referred to as *CAV12*, which was proposed and also considered in [8]. According
to [8], the benchmark constraints were generated by the program analysis tool
Compass (e.g. see [7]). In this setting, MSAs are important for reducing the
number of queries, which help users diagnose error reports as real bugs or false
alarms. Thus, the size of an MSA greatly affects the quality of queries presented
to users (and, hence, also the time spent on debugging users' programs). The
total number of variables for these instances varies from 1 to 53 and the total
number of instances in the benchmark set is 373.[7] The relative size of MSAs for
the CAV12 instances varies from 0 % (i.e. an instance is a tautology) to 100 %
(i.e. an MSA necessarily contains all variables) with the average size $\approx 58\,\%$.

The performance of the chosen competitors is shown in Fig. 1. Note that
the Y-axis of Fig. 1 is scaled logarithmically. As one can observe, the CAV12
instances are trivial to solve for all the competitors. All competitors spend about
10 s to solve the hardest instances in the benchmark suite. The averate running

---

[4] https://github.com/CVC4/CVC4.

[5] http://smtcomp.sourceforge.net/2015.

[6] Note that the original distribution of MISTRAL does not have a command-line
interface. But one can easily create one since the source code of the tool is available
online at https://www.cs.utexas.edu/~tdillig/mistral.

[7] We also tested the proposed approach on the standard SMTLIB benchmarks. How-
ever, minimum satisfying assignments for the majority of benchmarks in the QF_LIA
category of the SMTLIB benchmarks have trivial minimum satisfying assignments,
which contain all variables of the original formula. Therefore, considering these
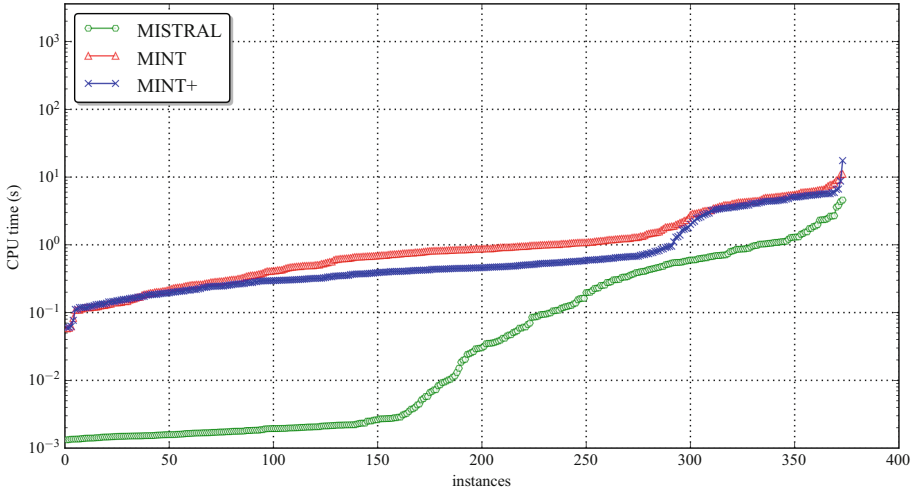instances makes no sense.

Fig. 1. Performance of MINT, MINT+, and MISTRAL on the CAV12 benchmark instances.

time for MINT, MINT+, and MISTRAL is 1.59 s, 1.23 s and 0.33 s, respectively. A possible explanation of why the new approach is about 0.1 second slower for most of the instances than MISTRAL (it takes 0.1 s vs 0.001 s spent on these instances by MISTRAL) is that it is implemented as a Python script, which requires some time to initialize the Python interpreter environment, while MISTRAL is run as a binary executable written in C++.

4.2 Hardened Benchmarks

The results for the CAV12 benchmark set suggest to consider harder instances in order to conduct a reasonable performance evaluation of the proposed approach. One way to create harder instances is to *combine* the existing CAV12 benchmarks with unsatisfiable formulas that are hard to solve, i.e. for each SMT formula $\mathcal{F}$ (from the CAV12 benchmark set) defined over variables X one needs to consider $\mathcal{F} \vee \mathcal{U}$, where $\mathcal{U}$ is an unsatisfiable formula over variables Y s.t. $X \cap Y = \emptyset$. Notice that, by construction, an MSA of formula $\mathcal{F}$ is also an MSA of $\mathcal{F} \vee \mathcal{U}$, and vice versa. As unsatisfiable components $\mathcal{U}$, we considered well-known families of unsatisfiable formulas, which are proved to be hard to refute by resolution-based reasoning, namely *pigeon-hole principle* formulas PHP_n [20] and formulas GT_n, which are based on the ordering principle that any partial order on a finite set must have a maximal element [4]. Given 373 CAV12 instances $\mathcal{F}$, the following experiments considered n ranging from 5 to 7 for both PHP_n and GT_n resulting in 3 combined benchmark sets $\mathcal{F} \vee PHP_n$ and 3 benchmark sets $\mathcal{F} \vee GT_n$, each also having 373 instances. The translation of CNF formulas PHP_n and GT_n was done in the standard way of encoding CNF formulas into integer linear

programming sets of constraints.[8] The number of Boolean variables in formulas PHP_n and GT_n is $n \times (n-1)$, which results in $2 \times n \times (n-1)$ integer variables appearing in the integer linear constraints encoding the original CNF formulas. Thus, the largest number of additional variables in the combination $\mathcal{F} \vee \mathcal{U}$ is 84 (for $n = 7$) and, hence, the largest total number of variables among the constructed benchmarks is 137.

The performance of the considered solvers shown for the $\mathcal{F} \vee PHP_n$ formulas is detailed in Fig. 2. The cactus plot shown in Fig. 2a illustrates how the performance of the competitors changes with the growth of n from 5 to 7. (Again, the Y-axis is scaled logarithmically in the cactus plot shown in Fig. 2a.) Observe that MINT+ and MINT are almost not affected by the unsatisfiable part of the formulas. However, the performance of MISTRAL drops significantly even for $n = 5$ (recall that for the CAV12 benchmarks the average running time of MISTRAL was 0.33 s while for $\mathcal{F} \vee PHP_5$ it is 149.4 s). This tendency towards increasing the running time dramatically for MISTRAL is persistent for $n \in \{6, 7\}$. The number of instances of $\mathcal{F} \vee PHP_6$ and $\mathcal{F} \vee PHP_7$ solved by MISTRAL is 195 and 9, respectively, whereas MINT+ can solve 373 and 371 and MINT solves 369 and 365 instances, respectively. The advantage of MINT+ over MISTRAL is confirmed by the scatter plot shown in Fig. 2b aggregating all instances $\mathcal{F} \vee PHP_n$, $n \in \{5, 6, 7\}$. As detailed in Fig. 3, similar results are shown by the new approach for the $\mathcal{F} \vee GT_n$ formulas while MISTRAL performs even worse (compared to

(a) Cactus plot   (b) MINT+ vs MISTRAL
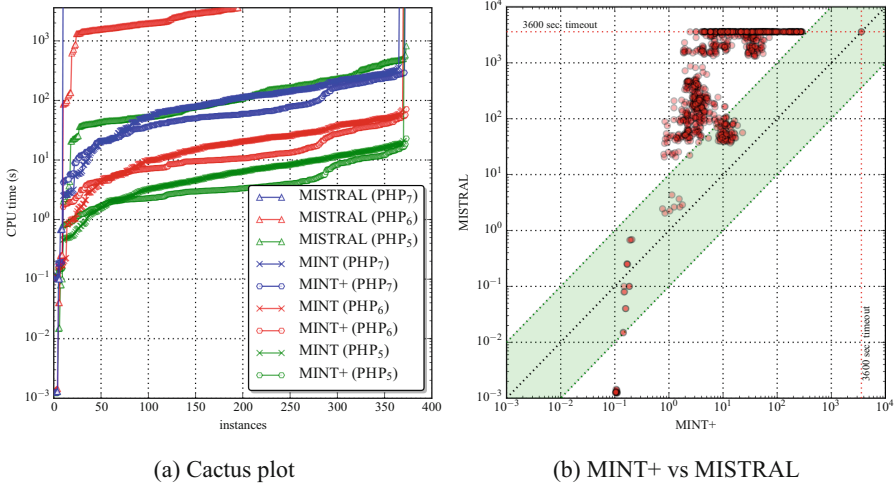
Fig. 2. Performance of MINT, MINT+, and MISTRAL on the $\mathcal{F} \vee PHP_n$ instances.

---

[8] For each variable x of the original CNF, two integer variables x_+ and x_- are introduced s.t. $0 \leq x_+ \leq 1$ and $0 \leq x_- \leq 1$. Variables x_+ and x_- cannot take value 0 or 1 at the same, which is forced by adding constraints of the form $x_+ + x_- = 1$. Each clause $l_1 \vee \ldots \vee l_m$ is translated into constraint $x_{1*} + \ldots + x_{m*} \geq 1$, where each x_{i*} represents either x_{i+} or x_{i-} depending on the polarity of literal l_i.

(a) Cactus plot (b) MINT+ vs MISTRAL
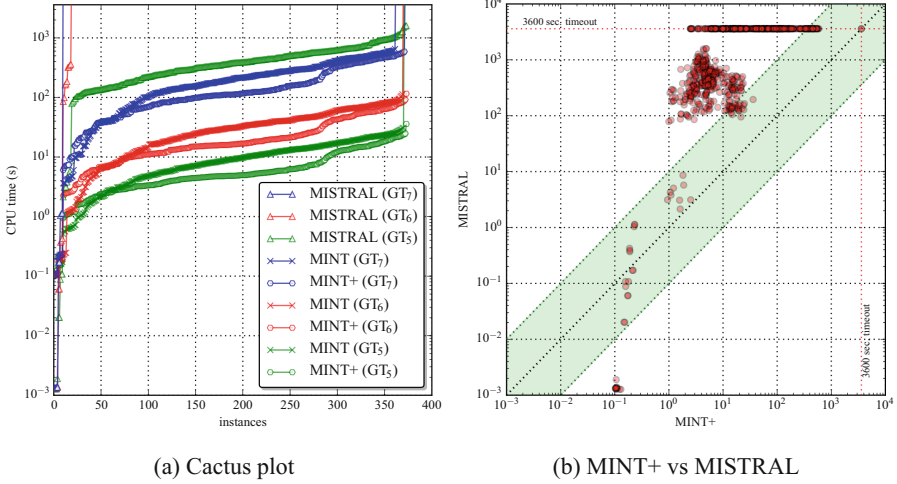
Fig. 3. Performance of MINT, MINT+, and MISTRAL on the $\mathcal{F} \vee GT_n$ instances.

195 $\mathcal{F} \vee PHP_6$ instances solved it solves only 18 instances of $\mathcal{F} \vee GT_6$). This confirms that for harder formulas the new approach significantly overperforms MISTRAL. This clear advantage of the new algorithm over MISTRAL is caused not only by the hardness of the PHP_n and GT_n formulas (since it does not affect the new approach that much) but also by a larger number of variables compared to the original CAV12 benchmark instances, which means that the proposed approach scales better in practice being able to solve harder problem instances.

5 Conclusions

MSAs [8] are generalizations of prime implicants for first-order logic formulas that find applications in a range of practical settings [7,15,24]. Recent work proposed a branch-and-bound approach for computing MSAs. In contrast, this paper proposes the use of an implicit hitting set solution [5,12–14,16,19,23] for computing MSAs. Experimental results, collected on challenging problem instances obtained from those used in earlier work [8], indicate significant performance gains compared to the earlier work [8].

The work described in this paper can be extended in different ways. First, more efficient algorithms for MSA will motivate additional applications and revisiting existing ones. Second, the growing range of uses of implicit hitting sets motivates devising more efficient algorithms.

References

1. Audemard, G., Lagniez, J.-M., Simon, L.: Improving glucose for incremental SAT solving with assumptions: application to MUS extraction. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 309–317. Springer, Heidelberg (2013)

2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011)

3. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, pp. 825–885. IOS Press (2009)

4. Beame, P., Kautz, H.A., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. J. Artif. Intell. Res. (JAIR) **22**, 319–351 (2004)

5. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 225–239. Springer, Heidelberg (2011)

6. de Moura, L., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)

7. Dillig, I., Dillig, T., Aiken, A.: Automated error diagnosis using abductive inference. In: PLDI, pp. 181–192 (2012)

8. Dillig, I., Dillig, T., McMillan, K.L., Aiken, A.: Minimum satisfying assignments for SMT. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 394–409. Springer, Heidelberg (2012)

9. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Heidelberg (2014)

10. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. **89**(4), 543–560 (2003)

11. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT Workshop (2015)

12. Ignatiev, A., Previti, A., Liffiton, M., Marques-Silva, J.: Smallest MUS extraction with minimal hitting set dualization. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 173–182. Springer, Heidelberg (2015)

13. Janota, M., Marques-Silva, J.: Solving QBF by clause selection. In: IJCAI, pp. 325–331 (2015)

14. Liffiton, M.H., Previti, A., Malik, A., Marques-Silva, J.: Fast, flexible MUS enumeration. Constraints **21**(2), 223–250 (2016)

15. Maity, S., Ghosh, S.K.: Conflict resolution in heterogeneous co-allied MANET: a formal approach. In: Chatterjee, M., Cao, J., Kothapalli, K., Rajsbaum, S. (eds.) ICDCN 2014. LNCS, vol. 8314, pp. 332–346. Springer, Heidelberg (2014)

16. Moreno-Centeno, E., Karp, R.M.: The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. OR **61**(2), 453–468 (2013)

17. Morgado, A., Ignatiev, A., Marques-Silva, J.: MSCG: robust core-guided MaxSAT solving. JSAT **9**, 129–134 (2015)

18. Palopoli, L., Pirri, F., Pizzuti, C.: Algorithms for selective enumeration of prime implicants. Artif. Intell. **111**(1–2), 41–72 (1999)

19. Previti, A., Ignatiev, A., Morgado, A., Marques-Silva, J.: Prime compilation of non-clausal formulae. In: IJCAI, pp. 1980–1987 (2015)

20. Razborov, A.A.: Proof complexity of pigeonhole principles. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 100–116. Springer, Heidelberg (2002)
21. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
22. Rymon, R.: An SE-tree-based prime implicant generation algorithm. Ann. Math. Artif. Intell. **11**(1–4), 351–366 (1994)
23. Saikko, P., Wallner, J.P., Järvisalo, M.: Implicit hitting set algorithms for reasoning beyond NP. In: KR (2016)
24. Vigo, R., Nielson, F., Nielson, H.R.: Uniform protection for multi-exposed targets. In: Ábrahám, E., Palamidessi, C. (eds.) FORTE 2014. LNCS, vol. 8461, pp. 182–198. Springer, Heidelberg (2014)

Towards a Dynamic Decomposition of CSPs with Separators of Bounded Size

Philippe Jégou, Hanan Kanso, and Cyril Terrioux[✉]

Aix-Marseille Université, CNRS, ENSAM, Université de Toulon,
LSIS UMR 7296, 13397 Marseille Cedex 20, France
{philippe.jegou,hanan.kanso,cyril.terrioux}@lsis.org

Abstract. In this paper, we address two key aspects of solving methods based on tree-decomposition. First, we propose an algorithm computing decompositions that allows to bound the size of separators, which is a crucial parameter to limit the space complexity, and thus the feasibility of such methods. Moreover, we show how it is possible to dynamically modify the considered decomposition during the search. This dynamic modification can offer more freedom to the variable ordering heuristics. This also allows to better use the information gained during the search while controlling the size of the required memory.

1 Introduction

The solving methods of CSPs based on tree-decomposition have shown a theoretical significance because they guarantee complexity bounds in $O(exp(w))$ in time as well as in $O(exp(s))$ in space where w and s are parameters induced by the structural properties of the constraint network. When w is bounded by a constant, these methods ensure a polynomial runtime. Moreover, in practice, such approaches are quite justified by numerous real-world problems for which w is relatively small [1]. However, two major problems occur sometimes in practice. First, controlling the value s is not always guaranteed, especially for decomposition methods like *Min-Fill* [2] which can be seen as the state of the art [3]. This sometimes makes this type of approach completely ineffective because this parameter is crucial in practice [4]. On the other hand, ensuring a time complexity in $O(exp(w))$ requires a traversal of the search space that imposes strong constraints on the variable assignment ordering, which can lead to a strong deterioration of practical efficiency.

To answer the question of memory, we propose a new configurable algorithm for computing decompositions. It takes as an input a parameter S to compute decompositions that guarantee separator sizes at most S. Its time complexity is less than that of *Min-Fill* and it offers performances which are widely better in practice (around 1,000 times faster on a large set of benchmarks). This algorithm fits perfectly into the framework proposed in [5] and can then be considered as a refinement of the heuristics proposed in this framework. The second part of the paper proposes a framework to dynamically change the decomposition

during the search, enabling to offer more freedom to heuristics while continuing exploiting decompositions. This approach relies on the fact that, to be efficient in practice, the solving methods must take into account the context of the search and the knowledge gained gradually during the search. This is done by CSP solvers using adaptive heuristics (e.g. [6,7]) and by CDCL SAT solvers (e.g. [8]) through clause learning and restart techniques. In the case of decomposition methods, the fundamental difficulty is linked to the variable ordering imposed by the decomposition. To overcome this difficulty, we propose to adapt dynamically the decomposition by merging clusters during the search. Such an approach have been introduced in [9] but mainly from a theoretical viewpoint. Thus, we show here how it is feasible. In addition, we extend it by integrating restarts techniques as proposed in [10]. Moreover, we describe how to dynamically change the decomposition, taking advantage of the knowledge acquired during the search while proposing to keep a bound on the size of separators all along the search. The last part of this paper presents an experimental analysis on a large set of instances, to assess the practical value of this approach.

In Sect. 2, we recall notions about solving methods based on tree-decompositions while in Sect. 3, we present the computation of tree-decompositions taking into account the size of separators. Section 4 introduces a variant of the algorithm BTD able to adapt the decomposition during search while Sect. 5 presents experiments that assess the relevance of this approach, before concluding.

2   Preliminaries

The *Constraint Satisfaction Problem* (CSP) provides a strong framework to formulate problems in computer science [11]. An instance of a finite CSP is given by a triple (X, D, C), with $X = \{x_1, \ldots, x_n\}$ a set of n variables, $D = \{d_{x_1}, \ldots, d_{x_n}\}$ a set of finite domains, and $C = \{c_1, \ldots, c_e\}$ a set of e constraints. Each constraint c_i is a pair $(S(c_i), R(c_i))$, where $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ defines the *scope* of c_i, and $R(c_i) \subseteq d_{x_{i_1}} \times \cdots \times d_{x_{i_k}}$ is its *compatibility relation*. The *arity* of c_i is $|S(c_i)|$. If the arity of each constraint is two, the instance is a *binary* CSP. The structure of a constraint network (other name of a CSP) is given by a hypergraph (a graph for a binary CSP), called the *constraint (hyper)graph*, whose vertices correspond to variables while edges correspond to the scopes of the constraints. To simplify notations, we denote the hypergraph $(X, \{S(c_1), \ldots S(c_e)\})$ by (X, C). An assignment on a subset of X is called *consistent* if all the constraints are satisfied. Checking whether a CSP has a *solution* (i.e. a consistent assignment of X) is well known to be NP-complete. So, many works have been done to improve the solving in practice such as algorithms exploiting heuristics, constraint learning, non-chronological backtracking or filtering-based algorithms. Nevertheless, the complexity of these approaches remains exponential, at least in $O(n.d^n)$ where d is the maximum size of domains. To circumvent this theoretical intractability, other approaches have been proposed. Some of them rely on a structural tractable class [12] based on the notion of *tree-decomposition of graphs* [13].

Definition 1. *A* tree-decomposition *of a graph* $G = (X, C)$ *is a pair* (E, T) *with* $T = (I, F)$ *a tree* (I *is the set of nodes and* F *the set of edges of* T) *and* $E = \{E_i : i \in I\}$ *a family of subsets of* X, *such that each subset (called cluster)* E_i *is a node of* T *and satisfies: (i)* $\cup_{i \in I} E_i = X$, *(ii) for each edge* $\{x, y\} \in C$, *there exists* $i \in I$ *with* $\{x, y\} \subseteq E_i$, *and (iii) for all* $i, j, k \in I$, *if* k *is in a path from* i *to* j *in* T, *then* $E_i \cap E_j \subseteq E_k$. *The width of a tree-decomposition* (E, T) *is equal to* $max_{i \in I}|E_i| - 1$. *The* tree-width w *of* G *is the minimal width over all the tree-decompositions of* G.

This notion is only defined for graphs but can be considered for a hypergraph by exploiting its *2-section*[1]. Their primary advantage is related to their theoretical time complexity in d^{w+1} [3] while their space complexity is in d^s where s is the maximum size of intersections (called *separators* in the sequel) between clusters. Thus, these methods can be efficient on large instances of small tree-width as it is the case for example for well known optimization problems of radio frequency allocations [15]. These methods run in two steps: (1) computing a tree-decomposition, and (2) solve the instance exploiting the decomposition. Since computing optimal decompositions (i.e. of width w) is NP-hard [16], in practice, the first step generally computes tree-decompositions whose width is $w^+ \geq w$, that is an approximation of the tree-width. In this context, *Min-Fill* [2] appears as the best compromise between the computation time ($O(n^3)$) and the quality of the obtained decompositions. It can be considered as the state of the art for such algorithms [3], even if, for graphs with more of tens of thousands of vertices, it may be unusable in practice.

However, the computed decompositions are not necessarily really suitable from a solving viewpoint [17,18]. First, *Min-Fill* does not take into account explicitly the topological properties of the considered graph which can make the solving inefficient. For example, the obtained decompositions may contain disconnected clusters [18]. Secondly, *Min-Fill* can generate decompositions such that s is often close to w^+. Indeed, in order to minimize the width, *Min-Fill* produces clusters with few proper vertices (i.e. vertices belonging to the cluster but not to its parent cluster in the tree-decomposition) or even only one proper vertex. This explains why s is often close to w^+. This can lead to a prohibitive cost for space memory. Thus, the minimization of s is crucial to be efficient in practice [17].

Secondly, to guarantee the time complexity in d^{w^+}, efficient structural methods such as BTD [19] use an ordering for the assignment of the variables which is partially determined by the considered decomposition. When *Min-Fill* is used, this freedom is even more restricted because of the limited number of proper vertices in the clusters. But we know that to have an efficient search, it is desirable to have maximum freedom when choosing the next variable to assign.

To circumvent these difficulties, several approaches are possible. A first approach is to have a decomposition with small separators, while having larger

---

[1] The 2-section of a hypergraph (X, C) is the graph (X, C') where $C' = \{\{x, y\}| \exists c \in C, \{x, y\} \subseteq c\}$ [14].

clusters thereby releasing the constraints on the ordering [17]. Another possibility is to exploit restarts like in [10]. This approach works by restarting the search from a first variable which does not necessarily belong to the previous root cluster. This leads, while retaining the same decomposition (except for the root cluster), to give more freedom to the ordering and its relevance has been shown experimentally. Another possibility is to dynamically change the decomposition during the search while maintaining guarantees for time complexity. This approach was proposed in [9], but mainly on a theoretical level. It consists in expanding the cluster size by merging some neighboring clusters. However, its relevance has never been demonstrated. Moreover, as defined in [9], it is only guided by structural criteria, without taking into account explicitly the state of search, and the knowledge gained during the solving. Note that exploiting the structure of instances dynamically has already been proposed for SAT in [20], but without guarantee for the complexity bounds, contrary to what we offer here.

To propose new alternative ways, we introduce in the following, first an algorithm which computes decompositions with clusters of bounded size. Secondly, we present a new solving algorithm based on BTD allowing to dynamically adapt the decompositions during the search, using information obtained from the beginning of the solving.

3   Decomposition Controlling Separators

3.1   A General Framework to Compute Specific Tree-Decompositions

In this part, we recall the framework *H-TD-WT* (*Heuristic Tree-Decomposition Without Triangulation* [5]) that computes a tree-decomposition of the graph $G = (X, C)$ without triangulation in polynomial time, more precisely in $O(n(n+e))$. Like Min-$Fill$, no warranty about the optimality of the computed width is given. However, it allows to compute decompositions depending on the features we want to fulfill. Notably, different parameterizations are conceivable depending on the wanted criteria for the obtained tree-decompositions. For example, these criteria may be related to w^+ and/or s or the connectivity of clusters [18]. By designing such a framework, we have many goals. First, in order to manage dynamically the decompositions during the solving, efficient decomposition algorithms are needed from a theoretical and practical viewpoint. Second, the complexity of these algorithms should be at most in $O(n(n+e))$ to be more efficient than Min-$Fill$. To do so, the time-consuming step of triangulation performed by Min-$Fill$ must be avoided. Beyond that, limiting the maximum size of the separators (i.e. intersection between clusters), as well as the size of clusters, is also crucial.

The first step of *H-TD-WT* (line 1 in Algorithm 1) computes a first cluster, denoted E_0, thanks to a heuristic. X' which denotes the set of already considered vertices is initialized to E_0 (line 2). We denote $X_1, X_2, \ldots X_k$ the connected components of the subgraph $G[X \backslash E_0]$ induced by the deletion in G

Algorithm 1. *H-TD-WT*

Input: A graph $G = (X, C)$
Output: A set of clusters $E_0, \ldots E_m$ of a tree-decomposition of G
1 Choose a first cluster E_0 in G
2 $X' \leftarrow E_0$
3 Let $X_1, \ldots X_k$ be the connected components of $G[X \backslash E_0]$
4 $F \leftarrow \{X_1, \ldots X_k\}$
5 **while** $F \neq \emptyset$ **do**   /* find new cluster E_i */
6   Delete X_i from F
7   Let $V_i \subseteq X'$ be the neighborhood of X_i in G
8   Find a subset $X_i'' \subseteq X_i$ such that there is at least one vertex $v \in V_i$ such that $N(v, X_i) \subseteq X_i''$
9 $E_i \leftarrow X_i'' \cup V_i$
10 $X' \leftarrow X' \cup X_i''$
11   Let $X_{i_1}, X_{i_2}, \ldots X_{i_{k_i}}$ be the connected components of $G[X_i \backslash E_i]$
12 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \ldots X_{i_{k_i}}\}$

---

of vertices of E_0[2]. Each one of these sets X_i is inserted in a queue F (line 4). For each element X_i deleted from F (line 6), V_i denotes the set of vertices of X' which are adjacent to at least one vertex of X_i (line 7). One can note that V_i is a separator in the graph G since removing V_i from G makes G disconnected (X_i being disconnected from the rest of G). We then consider the subgraph of G induced by V_i and X_i, that is $G[V_i \cup X_i]$. The next step (line 8) can be parameterized. It looks for a subset of vertices $X_i'' \subseteq X_i$ such that $X_i'' \cup V_i$ will be a new cluster E_i of the decomposition. This can be ensured if there is at least one vertex v of V_i s.t. all its neighbors in X_i appear in X_i''. More precisely, if $N(v, X_i) = \{x \in X_i : \{v, x\} \in C\}$, we must ensure that $\exists v, N(v, X_i) \subseteq X_i''$. We then define a new cluster $E_i = X_i'' \cup V_i$ (line 10). This process is repeated until the queue is empty. In [5], this framework implements several heuristics. The first (denoted H_1), tries to minimize the size of the clusters while the second (H_2) guarantees that clusters will be connected (see [18]). The third heuristic (H_3) aims to identify independent parts of the graph and to separate them as soon as possible using a breadth-first search starting from the vertices of V_i. The fourth heuristic (H_4), which introduces the one we will present in this contribution, aims to limit the size of the separators of the decomposition. To do so, it considers a parameter S which represents the maximum allowed size for a separator. This heuristic adds new vertices to the next cluster E_i similarly to H_3. Nevertheless, the heuristic stops progressing through levels at $l = L$ when $G[X_i \backslash E_{i_L}]$ does not contain any connected component with separator's size greater than S.

3.2 A Heuristic for Controlling Separators

We hereby introduce a new heuristic (denoted H_5) controlling the separator size. It aims to refine the heuristic H_4 by detecting more separators of size at most S. If H_4 stops adding vertices when it arrives to a level where all separators are of size at most S, H_5 may stop earlier. If at level l, the separator

---

[2] For any $Y \subseteq X$, the subgraph $G[Y]$ of $G = (X, C)$ induced by Y is the graph (Y, C_Y) where $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

associated to one of the connected components has at most size S, the separator will be taken into account and included in the obtained tree-decomposition. In fact, when the separator is detected the corresponding connected component is added to the queue in order to be managed later. Hence, the computation of the current cluster continues only on the remaining part of X_i after the removal of the connected component having a suitable separator. Consider the example given in Fig. 1(a). We first show the computation of E_1, the second cluster (after E_0) during the first pass through the loop and we set S to 2. We consider then the set $V_1 = \{x, y, z\}$. The vertices of the first level are then a, b and c. There is no subset of $\{a, b, c\}$ of size at most 2 that induces a separator of the graph. The next level that is visited contains the vertices d, e, f and g, At this level, we obtain two minimal separators, $\{d, e, f\}$ and $\{f, g\}$. If H_4 is used, the search would continue. However with H_5, the search is modified because of the detection and the exploitation of the separator $\{f, g\}$. Since $\{f, g\}$ is of size 2, the induced connected component containing the vertices i, j and m is removed and added to the queue F (line 8). Hence, the search continues only in the remaining part of the connected component X_1 which includes the vertices h, k, l, and n. The next level only contains the vertex h which is then a separator of size 1. Therefore, the search stops and a new cluster E_1 is created with: $E_1 = \{x, y, z, a, b, c, d, e, f, g, h\}$ and $X_1'' = \{a, b, c, d, e, f, g, h\}$. We obtain then a new connected component $X_{1_1} = \{k, l, n\}$ that is added to F. Note that the instantiation of $H\text{-}TD\text{-}WT$ by H_5 is integrated in line 8. The conditions required by the approach are thus respected. In particular, a new subset $X_i'' \subseteq X_i$ is created where there exists at least one vertex $v \in V_i$ with $N(v, X_i) \subseteq X_i''$. With this in mind and the proof given in [5], the validity of H_5 is ensured.
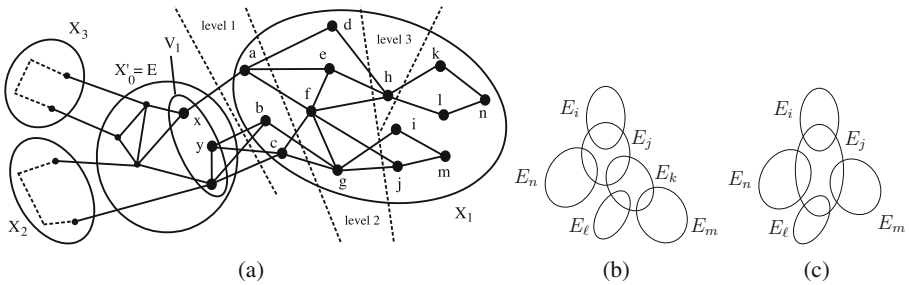

Fig. 1. View of $H5$ (a), a set of clusters of the decomposition before merging E_k with E_j (b) and after (c).

Theorem 1. H_5 *computes the clusters of a tree-decomposition.*

Also, the analysis of its complexity is similar to the one given in [5].

Theorem 2. *The time complexity of the algorithm H_5 is $O(n(n+e))$.*

4   The Dynamic Decomposition

4.1   Context

The thesis that we defend in this paper is that changing the decomposition dynamically during the solving allows to adapt the decomposition to the nature of the instance to solve. The decomposition is modified according to the knowledge acquired during the solving, especially the one related to the semantics of the problem. The approach can be thus classified among the *adaptive* methods. These methods make choices depending on the current state of the problem as well as previous states. In practice, they have shown their benefit (like in [6,7,21]) w.r.t. conventional methods. For example, with conflict-driven variable ordering heuristics, the most problematic variables are identified during the search thanks to learned information from the part of the problem already explored. Hence, this allows to consider the identified variables earlier in the search and so to solve the problem efficiently. Nevertheless, in the case of solving methods based on tree-decompositions like BTD, the variable ordering is partially induced by the used decomposition. In other words, the next chosen variable should be allowed by the tree-decomposition. For this reason, the structural methods suffer from the restrictions imposed by the tree-decomposition with regard to the variable ordering. In order to circumvent this problem, we propose to adapt the tree-decomposition during the solving by merging dynamically some clusters.

4.2   The Algorithm BTD-MAC+RST+Merge

The algorithm BTD-MAC+RST+Merge (see Algorithm 3) is an adaptation of the algorithm BTD-MAC+RST [10] in order to take into account the dynamic merging. For both algorithms, the use of a tree-decomposition having a root cluster E_r induces a partial variable ordering. If E_j is the current cluster, the freedom of variable ordering is limited to either choose among the unassigned variables of the cluster E_j or to choose the next cluster among the children of E_j when all its variables are assigned. Both algorithms compute first a tree-decomposition before the beginning of the solving. The main difference between them is that BTD-MAC+RST uses the initial decomposition during the solving (the same set of clusters but the root can change) while BTD-MAC+RST+Merge updates dynamically the tree-decomposition depending on the needs of the solving. Therefore, more different partial orderings can be exploited during the solving. The operation that permits to change the decomposition in this context is the *merging*. It consists in putting together the variables of two different clusters to create one cluster. Figure 1(c) shows the merging of two clusters E_j and E_k of the decomposition of Fig. 1(b). Note that, the children of the merged cluster become the children of the cluster resulting from the merging. For instance, E_ℓ and E_m, the children of E_k in Fig. 1(b), become the children of E_j in Fig. 1(c). Consider D the initial decomposition and D' the obtained decomposition after the merging. Any variable ordering allowed by D is also allowed by D'. Nonetheless, by exploiting D' we obtain more possible orderings

---

Algorithm 2. BTD-MAC+Merge (**InOut**: $P = (X, D, C)$: CSP; **In**: Σ: sequence of decisions, E_i: Cluster, V_{E_i}: set of variables; **InOut**: G: set of goods, N: set of nogoods)

---

1 **if** $V_{E_i} = \emptyset$ **then**
2       $result \leftarrow true$
3       $S \leftarrow Sons(E_i)$
4   **while** $result \notin \{false, unknown\}$ **and** $S \neq \emptyset$ **do**
5   Choose a cluster $E_j \in S$
6           $S \leftarrow S \backslash \{E_j\}$
7   **if** $Pos(\Sigma)[E_i \cap E_j]$ *is a nogood in* N **then** $result \leftarrow false$
8   **else**
9   **if** $Pos(\Sigma)[E_i \cap E_j]$ *is not a good of* E_i *w.r.t.* E_j *in* G **then**
10                  $result \leftarrow$BTD$(P, \Sigma, E_j, E_j \backslash (E_i \cap E_j), G, N)$
11   **if** $result = true$ **then** Record $Pos(\Sigma)[E_i \cap E_j]$ as good of E_i w.r.t. E_j in G
12   **else**
13   **if** $result = false$ **then** Record $Pos(\Sigma)[E_i \cap E_j]$ as nogood of E_i w.r.t. E_j in N
14   **else**
15   **if** *merge* **then** Merge E_j with one of its sons
16   **if** *not restart* **then**
17                              $S \leftarrow S \cup \{E_j\}$
18                              $result \leftarrow true$

19   **return** $result$
20 **else**
21   Choose a variable $x \in V_{E_i}$
22   Choose a value $v \in d_x$
23      $d_x \leftarrow d_x \backslash \{v\}$
24   **if** $AC (P, \Sigma \cup \langle x = v \rangle)$ **then** $result \leftarrow$ BTD$(P, \Sigma \cup \langle x = v \rangle, E_i, V_{E_i} \backslash \{x\}, G, N)$
25   **else** $result \leftarrow false$
26   **if** $result = false$ **then**
27   **if** *restart* **then**
28   Record nld-nogoods w.r.t. the decision sequence $(\Sigma \cup \langle x \neq v \rangle)[E_i]$
29   **return** $unknown$
30   **else**
31   **if** $AC (P, \Sigma \cup \langle x \neq v \rangle)$ **then return** BTD$(P, \Sigma \cup \langle x \neq v \rangle, E_i, V_{E_i}, G, N)$
32   **else return** $false$

33   **else return** result

---

than by using D. We then deduce that the merging preserves the orders initially allowed but also permits more freedom. Deciding to merge or not two clusters is only conditioned by the information learned during the solving. Also, the behavior of BTD-MAC+RST+Merge ranges from BTD-MAC+RST with a variable ordering partially imposed by the tree-decomposition (if no merging occurs) to MAC (for *Maintaining arc consistency* [22]) with a totally free variable ordering (if after several mergings, the decomposition contains only one cluster). So, the advantage of this new algorithm is its ability to find the right compromise thanks to learned information during the solving.

BTD-MAC+RST+Merge exploits the algorithm BTD-MAC+Merge (see Algorithm 2). The difference between BTD-MAC+Merge and BTD-MAC+NG [10] is located at lines 14–18. Initially, the sequence of decisions Σ as well as the set of goods G and nogoods N are empty. BTD-MAC+Merge

Algorithm 3. BTD-MAC+RST+Merge (**In**: $P = (X, D, C)$: CSP)

```
1  G ← ∅; N ← ∅
2  repeat
3       Choose a root cluster E_r
4       result ← BTD-MAC+Merge (P,∅,E_r,E_r,G,N)
5  until result ≠ unknown
6  return result
```

(like BTD-MAC+NG) begins the solving by assigning consistently the variables of the root cluster E_r before moving to one of its children. By exploiting the new cluster E_i, only unassigned variables of E_i are assigned. In other words, only the variables of E_i that do not belong to $E_i \cap E_{p(i)}$ (where $E_{p(i)}$ is the parent cluster of E_i) are assigned. In order to solve each cluster, both algorithms rely on MAC (lines 21–26 and 31–33). During the solving MAC can make two kind of decisions: positive decisions $x_i = v_i$ which assign the value v_i to the variable x_i and negative decisions $x_i \neq v_i$ which ensure that x_i cannot be assigned with v_i. Let us consider $\Sigma = \langle \delta_1, ..., \delta_i \rangle$ as the current decision sequence where each δ_j may be either a positive or a negative decision. A new positive decision $x_{i+1} = v_{i+1}$ is chosen and AC filtering is achieved (line 24). If no dead-end occurs, the search goes on by choosing a new positive decision (line 24). Otherwise, the value v_{i+1} is deleted from the domain $d_{x_{i+1}}$, and an AC filtering is realized (line 31). If a dead-end occurs again, we backtrack and change the last positive decision $x_\ell = v_\ell$ to $x_\ell \neq v_\ell$. When the cluster E_i is chosen as the next cluster, the next positive decision involves a variable of the current cluster E_i. Since $E_i \cap E_{p(i)}$ is a separator and all its variables are already assigned, only the domains of future variables in $Desc(E_i)$ are impacted by the AC filtering (where $Desc(E_i)$ is the set of variables belonging to the union of the descendants E_k of E_i). When the variables of the cluster E_i are consistently assigned (line 1), each subproblem rooted in each child cluster E_j of E_i is solved (line 10). More precisely, for a child E_j and a current decision sequence Σ, it attempts to solve the subproblem rooted in E_j induced by $Pos(\Sigma)[E_i \cap E_j]$ (where $Pos(\Sigma)[E_i \cap E_j]$ is the set of positive decisions involving the variables of $E_i \cap E_j$ in Σ). Once this subproblem solved, if a solution has been found by consistently extending Σ on $Desc(E_j)$ then $Pos(\Sigma)[E_i \cap E_j]$ is recorded as a *structural good*[3] (line 11). Otherwise if no solution exists, $Pos(\Sigma)[E_i \cap E_j]$ is recorded as a *structural nogood* (line 13). These structural (no)goods are exploited later during the solving to avoid redundancies (lines 7 and 9).

Regarding the restarts, they are managed like in [10]. If a restart occurs (line 27), the search is suspended and some reduced nld-nogoods [23] are recorded in order to avoid exploring again parts of the search tree already explored. The efficiency of restarts relies on the acquired knowledge and the exploitation of stored information via the structural (no)goods and the reduced

---

[3] A *structural good* (resp. *nogood*) of E_i w.r.t. E_j (with E_j a child of E_i) is a consistent assignment of $E_i \cap E_j$ which can (resp. cannot) be consistently extended on $Desc(E_j)$ [19].

nld-nogoods [23]. The restart condition may involve global parameters (related to the whole problem) or local parameters (related to the current cluster) or both. Lines 15–18 deal with the dynamic merging performed by BTD-MAC+Merge. The dynamic merging, as explained above in Sect. 2, aims to relax the constraints imposed by the decomposition on the variable ordering. Deciding to merge clusters or not depends on the current state of the problem as well as on all or part of its intermediate states. If no merging is required ($merge$ returns $false$), the search continues normally. On the contrary, if merging is judged relevant for the solving (e.g. by making possible the early assignment of important variables), $merge$ returns $true$ and BTD-MAC+Merge changes the current decomposition by merging the current cluster E_j with one of its children (line 15). To do so, all the assigned variables of $E_j \backslash (E_{p(j)} \cap E_j)$ are unassigned and the reduced nld-nogoods are recorded (line 28) as with conventional restarts. Once the search backtracks to the parent cluster, the merging is performed. In this case (line 16), either the backtracking through clusters continues if $restart$ returns $true$ or the search is resumed by exploring a child of the parent cluster. Note that, it is not mandatory to unassign the variables of the current cluster before merging. However, this would permit to consider the newly added variables in the cluster earlier in the search. BTD-MAC+Merge can be parameterized by a merging heuristic (we propose one in Sect. 5).

4.3   Theoretical Foundations

We now show the validity of our approach. First, we prove that the merging operation does not influence the validity of the structural (no)goods and the reduced nld-nogoods.

Proposition 1. *Let (E', T') be the tree-decomposition of the graph G obtained from the decomposition (E, T) of G after merging the cluster E_y with the cluster E_x (where E_y is a child of E_x in (E, T)). The recorded structural (no)goods of E_i w.r.t. E_j ($E_j \neq E_y$) and the recorded reduced nld-nogoods for (E, T) remain valid for (E', T').*

Proof: Consider Δ a structural good of E_i w.r.t. its child E_j recorded for (E, T). Knowing that E_j differs from E_y, the subproblem of P rooted in E_j in (E, T) is identical to the subproblem of P rooted in E_j in (E', T'). Hence, if Δ can be consistently extended on the first subproblem then it can also be extended on the second one. Consequently, Δ is a structural good of E_i w.r.t. E_j recorded for (E', T'). The reasoning is similar for a structural nogood. A reduced nld-nogood Δ is a nogood[4] whatever the considered tree-decomposition. We should only verify then if a nld-nogood Δ is valid for the decomposition (E', T'), that is to say that there exists a cluster of E' including all the variables of Δ. By construction,

---

[4] Given a CSP $P = (X, D, C)$ and a sequence of decisions Σ, Δ is a *nogood* of P if $P_{|\Delta}$ has no solution where $P_{|\Delta}$ is the CSP (X, D', C) with $D' = (d'_{x_1}, \ldots, d'_{x_n})$ and for each positive decision $x_i = v_i$, $d'_{x_i} = \{v_i\}$ and for each negative decision $x_i \neq v_i$, $d'_{x_i} = d_{x_i} \backslash \{v_i\}$. If x_i does not appear in Δ then $d'_{x_i} = d_{x_i}$ [23].

there exists necessarily a cluster E_k of (E, T) covering Δ. If $E_k \neq E_x$ and $E_k \neq E_y$ then $E_k \in E'$. Otherwise, after merging, we have $E_k \subset E_x$ and $E_x \in E'$. Therefore, in both cases, the variables of Δ are all covered by one cluster of E' and Δ is valid for (E', T'). $\qquad\square$

Then, we prove the validity of our algorithm.

Theorem 3. *BTD-MAC+RST+Merge is sound, complete and terminates.*

Proof: Consider first BTD-MAC+Merge which differs from BTD-MAC+NG by exploiting the merging. Assume that we obtain (E', T') from the decomposition (E, T) by merging two clusters. Let Σ_f be the sequence of decisions for which *merge* becomes *true*. Some reduced nld-nogoods are then recorded and the search backtracks to the parent cluster E_i of the current cluster E_j. Thus, we obtain the sequence Σ'_f that corresponds to the sequence of decisions Σ_f restricted to the variables of the clusters present in the branch going from the root cluster E_r to E_i. BTD-MAC+Merge continues the search from E_i with Σ'_f by exploiting the decomposition (E', T'). The cluster resulting from the merging can be the next visited cluster or can be visited later. The search tree explored by BTD-MAC+Merge between its first call with an empty sequence of decisions and the sequence of decisions Σ_f is the same as the one developed by BTD-MAC+NG under the same circumstances on the decomposition (E, T). Also, after the merging, the search tree developed by BTD-MAC+Merge between the sequence of decisions Σ'_f and its termination is identical to the one developed by BTD-MAC+NG under the same circumstances on the decomposition (E', T'). We know that BTD-MAC+NG is complete (if no restart occurs), correct and terminates [10]. Also, according to the Proposition 1, the structural (no)goods and the reduced nld-nogoods recorded for (E, T) remain valid for the new decomposition (E', T'). So, the correction, the termination and the completeness of the algorithm are not endangered. Furthermore, recording reduced nld-nogoods at each restart prevents from exploring a part of the search space already explored. Hence, BTD-MAC+Merge is complete (if no restart occurs), correct and terminates. In addition, when many merging operations are performed, the same reasoning can be applied for every merging by splitting the search tree. Note that, restarts stop the search without changing the fact that if a solution exists in the search space visited by BTD-MAC+Merge, BTD-MAC+Merge would find it. As BTD-MAC+RST+Merge only performs several calls to BTD-MAC+Merge, it is sound. Regarding the completeness, if the call to BTD-MAC+Merge is not stopped by a restart (what is necessarily the case of the last call to BTD-MAC+Merge if BTD-MAC+RST+Merge terminates), the completeness of BTD-MAC+Merge implies the one of BTD-MAC+RST+Merge. Furthermore, recording reduced nld-nogoods at each restart prevents from exploring a part of the search space already explored by a previous call to BTD-MAC+Merge. It ensues that, over successive calls to BTD-MAC+Merge, one has to explore a more and more reduced part of the search space. Hence, the termination and completeness of BTD-MAC+RST+Merge are ensured by the unlimited nogood recording achieved by the different calls

to BTD-MAC+Merge and by the termination and the completeness of BTD-MAC+Merge.   □

Finally, we give its time and space complexities.

Theorem 4. *BTD-MAC+RST+Merge has a time complexity in $O(R.((n.s^2 . e.\log(d) + w'^+.N).d^{w'^+ +2} + n.(w'^+)^2.d))$ and a space complexity in $O(n.s.d^s + w'^+.(d+N))$ with w'^+ the width of the final obtained decomposition, s the size of the largest intersection $E_i \cap E_j$ of the initial decomposition, R the number of restarts and N the number of recorded reduced nld-nogoods.*

Proof: BTD-MAC+RST has a time complexity in $O(((n.s^2.e.\log(d) + w^+.N).d^{w^+ +2} + n.(w^+)^2.d).R)$ and a space complexity in $O(n.s.d^s + w^+.(d+N))$ [10]. Regarding BTD-MAC+RST+Merge, applying the merging operations implies that the size of the clusters may increase. Hence, the theoretical complexities are expressed in terms of w'^+ instead of w^+. The merging operations do not create new clusters but, on the contrary, some are removed. Thus, the maximum size of separators in the initial decomposition represents an upper bound on the size of separators. Therefore, the time and space complexities of the elements related to the size of separators are not modified. Regarding the reduced nld-nogoods recorded after a merging operation, even though they induce additional time and space costs, these costs are already taken into account by the costs of recorded reduced nld-nogoods of restarts. Thereby, the time complexity is in $O(R.((n.s^2.e.\log(d) + w'^+.N).d^{w'^+ +2} + n.(w'^+)^2.d))$ and the space complexity is in $O(n.s.d^s + w'^+.(d+N))$.   □

Note that, we can limit the increase of the width of the obtained tree-decomposition regarding the width of the initial decomposition by using a suitable merging heuristic.

5   Experiments

In this section, we first present our experimental protocol before assessing H_5 w.r.t. the solving and comparing the dynamic decomposition with the static one and MAC+RST.

5.1   Experimental Protocol

Regarding the exploited tree-decompositions, we consider $Min\text{-}Fill$ (as the state of the art heuristic known for its good tree-width approximation), H_2 (which guarantees the connectivity of the clusters), H_3 (whose clusters have many children) and H_5 (which controls the size of the separators of the decomposition). We discard H_4 since it computes less elaborate decompositions than H_5. For H_2, the clusters are computed by choosing the vertices of the considered connected component by decreasing degree order until the cluster becomes connected (i.e. the heuristic $NV2$ of [18]). For H_5, the decomposition is exploited with different bounds on the size of separators.

The dynamic decomposition exploits a merging heuristic. This latter relies on the advices of the variable ordering heuristic to assess the need of clusters merging. More precisely, given a current cluster E_j, each time we choose the next variable to assign in E_j, we test whether the variable ordering heuristic would choose another variable if it has the opportunity to choose among the unassigned variables of $(\bigcup_{E_k \in Children(E_j)} E_k) \cup E_j$. If a variable of a child E_k of E_j is preferred to a variable of E_j, a counter related to E_k is incremented. When the counter related to E_k reaches the limit L (namely 100 in these experiments), the cluster E_k is merged with its parent E_j.

Regarding the solving, we consider BTD-MAC and BTD-MAC+RST as reference structural methods based on a static decomposition, BTD-MAC+RST+Merge and BTD-MAC+Merge (i.e. BTD-MAC+RST+Merge without restarts) for the methods exploiting dynamic decompositions, and MAC+RST as the reference conventional enumerative method. We choose as root cluster the cluster having the maximum ratio number of constraints to its size minus one. The arc-consistency is enforced by $AC3^{rm}$ for the preprocessing and $AC8^{rm}$ for the solving [24]. We use the heuristic dom/wdeg [6] to choose the next variable to assign and the geometric restart policy based on the number of performed backtracks with a ratio of 1.1 and an initial number of backtracks of 100.

All the algorithms are implemented in C++ in our own library. The experiments were performed on blade servers running Linux Ubuntu 14.04 each with two Intel Xeon processors E5-2609 v2 2.5 GHz and 32 GB of memory. We consider 1,859 CSP instances (the same as [5,10]) from the CSP 2008 competition[5]. Regarding the instances selection, we have excluded the instances having a trivial tree-decomposition (e.g. instances having a complete constraint graph) and the instances having global constraints (because global constraints are not taken into account yet by our CSP library). The solving is performed with a timeout of 15 min (including the computation of the decomposition).

5.2 H_5 vs Other Decompositions

Table 1 provides the number of solved instances and the cumulative runtime of each mentioned algorithm with each considered tree-decomposition. First, we compare the decomposition heuristics w.r.t. the solving efficiency of BTD-MAC. Regarding the number of solved instances, BTD-MAC with H_5 (with $S = 50$) solves the largest number of instances (namely 1,469) while with $Min\text{-}Fill$, it solves the least number of instances (namely 1,344). Clearly, decompositions aiming to minimize the width are not necessarily the most efficient w.r.t. the solving. Other parameters have more impact on the solving such as the connectivity of the clusters (with H_2), the number of children of a cluster (with H_3) as well as the maximum size of separators (with H_5). Note that these results are consistent with ones of [5,18]. Besides, of course, with H_5, the efficiency of the solving depends on the chosen value for S. For instance, BTD-MAC solves more

Table 1. Number of solved instances and runtime for BTD-MAC, BTD-MAC+RST, BTD-MAC+Merge and BTD-MAC+RST+Merge depending on the exploited decompositions.

Algorithm	Min-Fill		H_2		H_3		H_5 $(S = 50)$	
	#solved	time	#solved	time	#solved	time	#solved	time
BTD-MAC	1,344	43,272	1,405	31,429	1,466	31,469	1,469	33,564
BTD-MAC+RST	1,495	43,557	1,518	35,042	1,529	30,187	1,543	33,049
BTD-MAC+Merge	1,481	42,505	1,518	37,440	1,523	35,101	1,534	34,048
BTD-MAC+RST+Merge	1,544	41,622	1,547	32,547	1,554	33,736	1,567	34,432

Table 2. Runtime for BTD-MAC, BTD-MAC+RST, BTD-MAC+Merge and BTD-MAC+RST+Merge depending on the exploited decompositions for the 1,234 instances solved by all the algorithms.

Algorithm	Min-Fill	H_2	H_3	H_5 $(S = 50)$
BTD-MAC	34,669	18,018	18,951	18,243
BTD-MAC+RST	24,026	17,233	17,758	16,288
BTD-MAC+Merge	25,238	17,575	18,753	17,837
BTD-MAC+RST+Merge	23,832	16,803	17,602	15,718

instances with $S = 15$ (namely 1,514). Choosing $S = 50$ is more interesting for exploiting dynamic decompositions.

Regarding the runtime, for a fair comparison, we consider, in Table 2, the 1,234 instances solved by all the algorithms. Again, BTD-MAC obtains the best results with H_5 (and H_2) and the worst ones with $Min\text{-}Fill$. We must point out that computing tree-decompositions thanks to H_i $(i = 2, 3, 5)$ is significantly faster than with $Min\text{-}Fill$. For instance, H_5 (with $S = 50$) only requires 7 s to compute the tree-decompositions for all the 1,234 instances while $Min\text{-}Fill$ needs 7,582 s.

Note that the benefits of H_5 observed here for BTD-MAC are valid whatever the variant of BTD we use as shown in Tables 1, 2, 3 and 4.

5.3 Dynamic Decompositions vs Static Decompositions

First, if we compare BTD-MAC to BTD-MAC+Merge (resp. BTD-MAC+RST to BTD-MAC+RST+Merge) in Tables 1 and 2, whatever the used decomposition, we can observe that the methods exploiting dynamic decompositions solve more instances than their corresponding variants exploiting a static decomposition while their runtime is either similar or better. This clearly highlights the benefits of dynamically merging clusters during the solving.

Nevertheless, the concept of merging may also be performed statically, as advocated in [17], after having computed first a tree-decomposition thanks to any algorithm (e.g. $Min\text{-}Fill$, H_2, H_3 and even H_5). Tables 3 and 4 provide

Table 3. Number of solved instances and runtime for BTD-MAC and BTD-MAC+RST depending on the exploited decompositions with a static merging limiting the size of separators to 15.

Algorithm	Min-$Fill$		H_2		H_3		H_5 $(S = 50)$	
	#solved	time	#solved	time	#solved	time	#solved	time
BTD-MAC	1,450	45,988	1,493	35,871	1,504	31,612	1,511	32,097
BTD-MAC+RST	1,537	41,722	1,549	33,328	1,553	33,164	1,564	33,145

Table 4. Runtime for BTD-MAC and BTD-MAC+RST depending on the exploited decompositions for the 1,234 instances solved by all the algorithms.

Algorithm	Min-$Fill$	H_2	H_3	H_5 $(S = 50)$
BTD-MAC	32,641	17,914	17,813	16,503
BTD-MAC+RST	24,456	17,514	17,050	16,235

the corresponding results for BTD-MAC(+RST). Here, we limit the size of the separators by merging with its parent any cluster whose separator with its parent exceeds a given value (namely 15 in Tables 3 and 4). We can note that, regarding the number of solved instances, BTD-MAC+Merge is significantly better than BTD-MAC while BTD-MAC+RST+Merge is comparable or slightly better than BTD-MAC+RST. Again, the exploitation of dynamic decomposition leads to obtain one of the best results. Beyond, by dynamically merging some clusters during the solving, we adapt the decomposition depending on some semantic knowledge about the instance whereas the static merging relies only on structural criteria and requires to choose a limit for the separator size, what may be a difficult task.

Finally, we can remark that BTD-MAC+RST and BTD-MAC+Merge are relatively close w.r.t. the number of solved instances or the runtime. This can be explained by the choice of a new root cluster when BTD-MAC+RST restarts, what can be seen as a light form of dynamicity for the decomposition. Moreover, the exploitation of both restarts and dynamic decompositions is really relevant since BTD-MAC+RST+Merge outperforms both BTD-MAC+RST and BTD-MAC+Merge. At the end, we can note that BTD-MAC+RST+Merge with H_5 obtain the best results whatever the decomposition or the solving algorithm we use.

5.4   BTD-MAC+RST+Merge Versus MAC+RST

We now compare BTD-MAC+RST+Merge versus MAC+RST w.r.t. the solving efficiency. For this, we consider here $S = 50$. Figure 2(a) presents the cumulative number of solved instances for MAC-BTD+RST+Merge, MAC+RST and VBS (i.e. the Virtual Best Solver among the two algorithms). First, BTD-MAC+RST+Merge solves more instances than MAC+RST (1,567 instances
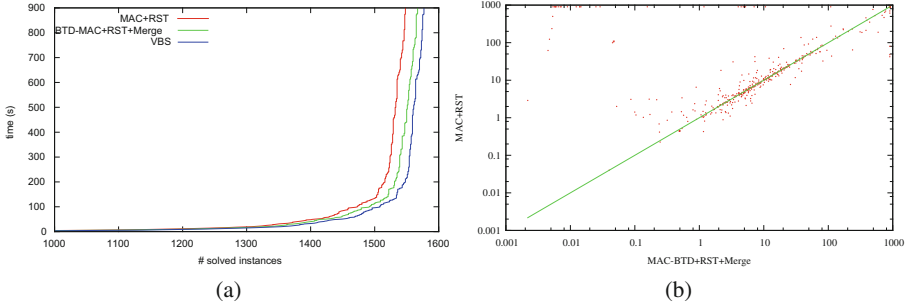
Fig. 2. (a) The cumulative number of solved instances for MAC-BTD+RST+Merge with H_5 (S = 50), MAC+RST and VBS, (b) runtime comparison for MAC-BTD+RST+Merge and MAC+RST for the 577 difficult instances.

against 1,548). Then, we can note that the behavior of BTD-MAC+RST+Merge is closer to one of VBS than one of MAC+RST, what clearly shows that BTD-MAC+RST+Merge performs better than MAC+RST.

Now we focus our observations on the hardest instances. Among the 1,859 considered instances, some of them are easily solved by MAC+RST (e.g. 284 instances are solved in backtrack-free manner). Exploiting structural methods like BTD or its variants for solving such instances is not necessarily relevant. So, we exploit here the number of nodes developed by MAC+RST as a hardness criterion. An instance is considered as difficult if the number of nodes developed by MAC+RST is greater than $100n$ (with n the number of variables). By so doing, we have 577 instances considered as difficult. Figure 2(b) provides a runtime comparison for MAC-BTD+RST+Merge and MAC+RST for these instances. Globally, we can observe that MAC-BTD+RST+Merge and MAC+RST have a similar behavior on a large part of these instances. Indeed, for about 60 % of the instances, the runtime gap between the two methods is less than 10 %. However, for the remaining instances, MAC-BTD+RST+Merge often outperforms MAC+RST. For 16 % of them, MAC-BTD+RST+Merge is at least 10 times faster than MAC+RST while MAC+RST performs 10 times faster for only 1 %. Finally, the exploitation of the structure plays here a central role. Indeed, we can note that 86 % of the instances unsolved by MAC+RST but solved by BTD-MAC+RST+Merge are structured instances having a ratio $n/(w+1)$ greater than 5.

6   Conclusion

In this paper, we proposed two complementary contributions. On the one hand, we presented a new algorithm for computing tree-decompositions (namely H_5) allowing us to bound the size of separators, which is a crucial parameter for the practical efficiency of structural solving methods like BTD. Its time complexity is better than the one of *Min-Fill* and it runs about 1,000 times faster than

Min-Fill on a large set of instances. On the other hand, we described a non straightforward extension of BTD, namely BTD-MAC+RST+Merge, which has the ability of adapting the tree-decomposition by dynamically merging some clusters depending on the semantics of the instance and the knowledge acquired during the solving. By so doing, our method exploits more flexible variable orderings and may correct some drawbacks of the initial tree-decomposition whose computation relies only on structural parameters. In practice, we showed that BTD-MAC+RST+Merge outperforms BTD-MAC+RST whatever the exploited decompositions. Moreover, its use jointly with H_5 leads to obtain the best results.

For future investigations, first, other merging heuristics are possible by exploiting different information related to the semantics learned during the solving. Then, the fact that H-TD-WT is much more faster than *Min-Fill* allows to compute more elaborate decompositions during the solving and on restarts. Beyond, more difficult problems can be tackled (e.g. optimization, counting or compilation).

References

1. de Givry, S., Schiex, T., Verfaillie, G.: Exploiting tree decomposition and soft local consistency in weighted CSP. In Proceedings of AAAI, pp. 22–27 (2006)
2. Rose, D.J.: A graph theoretic study of the numerical solution of sparse positive definite systems of linear equations. In: Graph Theory and Computing, pp. 183–217. Academic Press (1972)
3. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers, San Francisco (2003)
4. Allouche, D., de Givry, S., Schiex, T.: Towards parallel non serial dynamic programming for solving hard weighted CSP. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 53–60. Springer, Heidelberg (2010)
5. Jégou, P., Kanso, H., Terrioux, C.: An algorithmic framework for decomposing constraint networks. In: Proceedings of ICTAI, pp. 1–8 (2015)
6. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of ECAI, pp. 146–150 (2004)
7. Refalo, P.: Impact-based search strategies for constraint programming. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 557–571. Springer, Heidelberg (2004)
8. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
9. Jégou, P., Ndiaye, S.N., Terrioux, C.: Dynamic management of heuristics for solving structured CSPs. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 364–378. Springer, Heidelberg (2007)
10. Jégou, P., Terrioux, C.: Combining restarts, nogoods and decompositions for solving CSPs. In: Proceedings of ECAI, pp. 465–470 (2014)
11. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier, New York (2006)
12. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Artif. Intell. **124**, 243–282 (2000)
13. Robertson, N., Seymour, P.D.: Graph minors II: algorithmic aspects of treewidth. Algorithms **7**, 309–322 (1986)
14. Berge, C.: Graphs and Hypergraphs. Elsevier, New York (1973)

15. Cabon, C., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P.: Radio link frequency. Constraints **4**, 79–89 (1999)
16. Arnborg, S., Corneil, D., Proskuroswki, A.: Complexity of finding embeddings in a k-tree. SIAM J. Disc. Math. **8**, 277–284 (1987)
17. Jégou, P., Ndiaye, S.N., Terrioux, C.: Computing and exploiting tree-decompositions for solving constraint networks. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 777–781. Springer, Heidelberg (2005)
18. Jégou, P., Terrioux, C.: Tree-decompositions with connected clusters for solving constraint networks. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 407–423. Springer, Heidelberg (2014)
19. Jégou, P., Terrioux, C.: Hybrid backtracking bounded by tree-decomposition of constraint networks. Artif. Intell. **146**, 43–75 (2003)
20. Li, W., van Beek, P.: Guiding Real-World SAT solving with dynamic hypergraph separator decomposition. In: Proceedings of ICTAI, pp. 542–548 (2004)
21. Michel, L., Van Hentenryck, P.: Activity-based search for black-box constraint programming solvers. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 228–243. Springer, Heidelberg (2012)
22. Sabin, D., Freuder, E.: Contradicting conventional wisdom in constraint satisfaction. In: Borning, A. (ed.) PPCP 1994. LNCS, vol. 874, pp. 125–129. Springer, Heidelberg (1994)
23. Lecoutre, C., Saïs, L., Tabary, S., Vidal, V.: Recording and minimizing nogoods from restarts. JSAT **1**(3–4), 147–167 (2007)
24. Lecoutre, C., Likitvivatanavong, C., Shannon, S., Yap, R., Zhang, Y.: Maintaining arc consistency with multiple residues. Constraint Program. Lett. **2**, 3–19 (2008)

Constraint Programming for Strictly Convex Integer Quadratically-Constrained Problems

Wen-Yang Ku$^{(\boxtimes)}$ and J. Christopher Beck

Department of Mechanical and Industrial Engineering, University of Toronto,
Toronto, ON M5S 3G8, Canada
{wku,jcb}@mie.utoronto.ca

Abstract. Inspired by the geometric reasoning exploited in discrete ellipsoid-based search (DEBS) from the communications literature, we develop a constraint programming (CP) approach to solve problems with strictly convex quadratic constraints. Such constraints appear in numerous applications such as modelling the ground-to-satellite distance in global positioning systems and evaluating the efficiency of a schedule with respect to quadratic objective functions. We strengthen the key aspects of the DEBS approach and implement them as combination of a global constraint and variable/value ordering heuristics in IBM ILOG CP Optimizer. Experiments on a variety of benchmark instances show significant improvement compared to the default settings and state-of-the-art performance compared to competing technologies of mixed integer programming, semi-definite programming, and mixed integer nonlinear programming.

1 Introduction

The strictly convex integer quadratically-constrained problem (IQCP) is an optimization problem where the objective and/or some constraints are strictly convex quadratic functions. The IQCP is known to be NP-hard [1] and arises in a number of applications including global positioning systems, communications, cryptography, bioinformatics, scheduling and finance [2–5]. Given its theoretical challenge and practical value, it is of great interest to develop efficient algorithms to solve IQCPs. Despite the long history of CP, the techniques to solve quadratically constrained problems have not receive much attention. There are only a few dedicated global constraints that reason about quadratic terms. For example, the SPREAD constraint [6] enforces the standard quadratic relationship amongst a set of variables, their mean, and their standard deviation. General quadratic constraints [7,8] can be applied to both convex and nonconvex quadratic functions. However, they do not exploit the strictly convex nature of the IQCP.

It has been shown in Ku and Beck [9] that strictly convex IQCPs can be formulated as integer least squares (ILS) problems and solved with discrete ellipsoid-based search (DEBS), a specialized search used in the communications literature (e.g., see [4]). From a CP perspective, DEBS can be understood as a form of CP search. First, the search strategy uses a static variable ordering

heuristic and a dynamic value ordering heuristic based on the structure of the ellipsoid. Second, the geometry of the ellipsoid induces an interval domain for each variable. As a result, DEBS is essentially the enumeration of these domains under the prescribed variable and value orderings with some bounds pruning based on the radius of the hyper-ellipsoid. DEBS was originally formulated to solve only three types of ILS problems: unconstrained, box-constrained, and ellipsoid-constrained [3,9–11]. A recent work has extended DEBS for ILS problems with general linear constraints [12].

In this work, we aim at developing techniques that are both powerful and cover a broad range of problems. We introduce two novel techniques, inspired by DEBS, for solving strictly convex IQCPs with constraint programming. First, we propose the ELLIPSOID constraint, a global constraint that filters variable domains with respect to strictly convex quadratic functions. We derive a direct quadratically-constrained programming (QCP) formulation that achieves bounds consistency (BC), and two light-weight filtering algorithms that do not guarantee BC. Though it is natural to consider integer domains in CP, our filtering algorithm can be applied to variables with real domains, broadening its application to, for example, mixed integer programming solvers. Second, we propose a pair of variable/value selection rules. We implement the filtering algorithms and the branching heuristics in IBM ILOG CP Optimizer. We experiment with five problem classes and show orders of magnitude improvement compared to the default CP Optimizer. We then compare our new CP approach with the best known algorithms on the same problem sets. Our results demonstrate that the new CP approach is competitive to the best known approaches, and, for some problem classes, establishes a new state of the art.

The rest of the paper is organized as follows. We give the necessary background in Sect. 2. In Sect. 3 we define of our new global constraint. Sections 4 and 5 present the filtering algorithms and the branching heuristics. Section 6 provides computational results and discussions. We conclude in Sect. 7.

2   Background

2.1   The Strictly Convex Integer Quadratically-Constrained Problem (IQCP)

The general IQCP problem has the following form:

$$\min_{\boldsymbol{x} \in \mathcal{C}} \frac{1}{2} \boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} + \boldsymbol{f}^T \boldsymbol{x},$$

$$\mathcal{C} = \{\boldsymbol{x} \in \mathbb{Z}^n : \frac{1}{2} \boldsymbol{x}^T \boldsymbol{M}_k \boldsymbol{x} + \boldsymbol{c}_k^T \boldsymbol{x} \le \boldsymbol{b}_k, \forall k = 1, \dots, m, \ \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}, \ \boldsymbol{l} \in \mathbb{Z}^n, \ \boldsymbol{u} \in \mathbb{Z}^n\}.$$

The IQCP is strictly convex if the quadratic matrices $\boldsymbol{H}$, $\boldsymbol{M}_k, \forall k$ are symmetric positive definite [13]. As the above form suggests, quadratic formulations can exist in the form of an objective function and/or as constraints.

In Operations Research, the common generic approaches to solving IQCPs exactly are the use of mixed integer nonlinear programming (MINLP) such as BARON [14,15] and ANTIGONE [16], and the application of MIP solvers such as CPLEX and Gurobi which have been extended to reason about quadratic constraints [17]. Another generic approach is semi-definite programming (SDP) based branch-and-bound [18]. The available SDP solvers, e.g., BiqCrunch [19], only solve problems with binary variables as opposed to general integer variables.

2.2   Discrete Ellipsoid-Based Search (DEBS)

The DEBS method consists of two phases: reduction and search. The reduction is a preprocessing step that transforms $\boldsymbol{A}$ to an upper triangular matrix $\boldsymbol{R}$ using the "QRZ" factorization [3]:

$$\min_{\boldsymbol{x}\in\mathbb{Z}^n} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \rightarrow \min_{\boldsymbol{z}\in\mathbb{Z}^n} \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2, \tag{1}$$

where $\bar{\boldsymbol{y}} = \boldsymbol{Q}^T\boldsymbol{y}$, $\boldsymbol{z} = \boldsymbol{Z}^{-1}\boldsymbol{x}$, $\boldsymbol{Q}$ is orthogonal, $\boldsymbol{Z}$ is unimodular. The diagonal entries of $\boldsymbol{R}$ are approximately[1] ordered in non-decreasing order: $|r_{ii}| \leq |r_{i+1,i+1}|$. This ordering has been shown to increase the efficiency of the DEBS search by reducing the branching factor at the top of the search tree [10]. As we argue in Sect. 5 below, this ordering is approximately equivalent to a static smallest domain first variable ordering.

Suppose the optimal solution $\boldsymbol{z}^*$ satisfies $\|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}^*\|_2^2 < \beta$, or equivalently $\sum_{k=1}^n (\bar{y_k} - \sum_{j=k}^n r_{kj}z_j)^2 < \beta$, where β is a constant that can be obtained by substituting any feasible integer solution to Eq. (1). This expression defines a hyper-ellipsoid with center $\boldsymbol{R}^{-1}\bar{\boldsymbol{y}}$. The search, then, systematically enumerates all the integer points in the bounded hyper-ellipsoid [21]. When an incumbent, i.e., new upper bound on β, is found, the hyper-ellipsoid is contracted resulting in reduction of the bounds of the decision variables.

In more detail, let $\boldsymbol{z}_i^n = [z_i, z_{i+1}, \ldots, z_n]^T$ be the vector of decision variables and define the so-far-unknown (apart from c_n) and usually non-integer variables:

$$c_n = \bar{y}_n/r_{nn}, \quad c_k : c_k(z_{k+1}, \ldots, z_n) = (\bar{y}_k - \sum_{j=k+1}^n r_{kj}z_j)/r_{kk}, \quad k = n-1, \ldots, 1.$$

Note that c_k is a function of z_{k+1} to z_n, and it is fixed when z_{k+1} to z_n are fixed. The above equation can be rewritten as $\sum_{k=1}^n r_{kk}^2(z_k - c_k)^2 < \beta$, which defines the possible values that z_k can take on. This inequality is equivalent to the following n inequalities:

$$\text{level } n: \quad (z_n - c_n)^2 < \frac{1}{r_{nn}^2}\beta,$$

$$\text{level } n-1: \quad (z_{n-1} - c_{n-1})^2 < \frac{1}{r_{n-1,n-1}^2}[\beta - r_{nn}^2(z_n - c_n)^2],$$

---

[1] Depending on the data, it is sometimes not possible to transform a matrix to exactly achieve this ordering [20].

$$\vdots$$

$$\text{level } k: \quad (z_k - c_k)^2 < \frac{1}{r_{kk}^2}[\beta - \sum_{i=k+1}^{n} r_{ii}^2(z_i - c_i)^2],$$

$$\vdots$$

$$\text{level } 1: \quad (z_1 - c_1)^2 < \frac{1}{r_{11}^2}[\beta - \sum_{i=2}^{n} r_{ii}^2(z_i - c_i)^2].$$

The search starts at level n, heuristically assigning $z_n = \lfloor c_n \rceil$, the nearest integer to c_n. Given the value of z_n, c_{n-1} can be calculated from the above equation as $c_{n-1} = (\bar{y}_{n-1} - r_{n-1,n}z_n)/r_{n-1,n-1}$. From this value, we can set $z_{n-1} = \lfloor c_{n-1} \rceil$ and search continues. During the search process, z_k is determined at level k, where $z_n, z_{n-1}, \ldots, z_{k+1}$ have already been determined, but $z_{k-1}, z_{k-2}, \ldots, z_1$ are still unassigned. At some level $k - 1$ in the search, it is likely that the inequality cannot be satisfied, requiring the search to backtrack to a previous decision. When we backtrack from level $k - 1$ to level k, we choose z_k to be the next nearest integer to c_k.

After the optimal solution $\boldsymbol{z}^*$ to the reduced problem (right hand side of Eq. (1)) is found, the optimal solution, $\boldsymbol{x}^*$, to the original problem (left hand side of Eq. (1)) can be recovered with the relationship $\boldsymbol{x}^* = \boldsymbol{Z}\boldsymbol{z}^*$.

3   The Ellipsoid Constraint

We propose the ELLIPSOID constraint to reason about convex quadratic functions. It consists of a set of n variables $\{x_1, \ldots, x_n\}$, an $n \times n$ matrix $\boldsymbol{A}$ with full column rank, an n-dimensional vector $\boldsymbol{y}$, and a constant β. The definition is given as follows:

$$\texttt{ellipsoid}(\{x_1, \ldots, x_n\}, \boldsymbol{A}, \boldsymbol{y}, \beta),$$

where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, $\boldsymbol{y} \in \mathbb{R}^n$, $\beta \in \mathbb{R}$. The constraint ensures the following condition:

$$\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \leq \beta. \tag{2}$$

Geometrically, the above expression defines a hyper-ellipsoid with center $\boldsymbol{A}^{-1}\boldsymbol{y}$. Equivalently, (2) can be written in its standard convex quadratic constraint form as

$$\frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^T\boldsymbol{x} \leq \bar{\beta}, \tag{3}$$

where $\boldsymbol{H} \in \mathbb{R}^{n \times n}$ is a symmetric positive definite matrix, $\boldsymbol{f} \in \mathbb{R}^n$ is a vector, and $\bar{\beta} = (\beta - \boldsymbol{y}^T\boldsymbol{y})/2$. The transformation is obtained with the relationships $\boldsymbol{H} = \boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{f} = -\boldsymbol{y}^T\boldsymbol{A}$.

The ELLIPSOID constraint can be applied to any formulation with a strictly convex quadratic function. For example, consider the following objective function: $\min \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{y}^T\boldsymbol{H}_1\boldsymbol{y} + \boldsymbol{f}_1^T\boldsymbol{y}$, where only $\boldsymbol{H}$ is symmetric positive definite. We can still apply the ELLIPSOID constraint to the first half of the objective function: $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^T\boldsymbol{x}$, even if the second part is not strictly convex.

4 Filtering Algorithms for the Ellipsoid Constraint

In this section, we present a number of filtering algorithms that achieve or approximate bounds consistency of the ELLIPSOID constraint.

Let x_j be a finite-domain variable, $Dom(x_j)$ be the domain of x_j, which is a set of ordered values that can be assigned to x_j, and $I_D(x_j) = [l_j, u_j]$ be the interval domain of x_j.

Definition 1. *A* ELLIPSOID *constraint is bounds consistent [22] with respect to domains* $Dom(x_j)$ *if for all* $j \in 1, \ldots, n$ *and each value* $v_j \in \{l_j, u_j\}$*, there exists values* $v_i \in I_D(x_i)$ *for all* $i \in \{1, \ldots, n\} \setminus \{j\}$ *such that* `ellipsoid`$(\{x_1 = v_1, \ldots, x_n = v_n\}, \boldsymbol{A}, \boldsymbol{y}, \beta)$ *holds.*

In the 2D example shown in Fig. 1, the ELLIPSOID constraint (on the two variables) is bounds consistent.
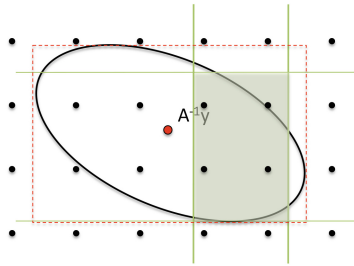

Fig. 1. A 2D example that shows the tangent box of the ellipsoid and bounds consistency of the ELLIPSOID constraint.

4.1 A Direct Quadratically-Constrained Programming (QCP) Formulation

Achieving bounds consistency for a variable x_j is equivalent to finding the lower bound l_j^{BC} and upper bound u_j^{BC} for x_j, given the ELLIPSOID constraint and the current bounds of the variables. Assume that no further reduction can be inferred on the domains of $x_i, \forall i \neq j$, the mathematical model for achieving bounds consistency for x_j can be defined as follows:

$$l_j^{BC} = \min_{\boldsymbol{x} \in \mathbb{R}^n} \boldsymbol{e}_j^T \boldsymbol{x} \quad \text{subject to } \|\boldsymbol{y} - \boldsymbol{Ax}\|_2 \leq \sqrt{\beta},\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \qquad (4)$$

$$u_j^{BC} = \max_{\boldsymbol{x} \in \mathbb{R}^n} \boldsymbol{e}_j^T \boldsymbol{x} \quad \text{subject to } \|\boldsymbol{y} - \boldsymbol{Ax}\|_2 \leq \sqrt{\beta},\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}. \qquad (5)$$

Note that $\boldsymbol{e}_j \in \mathbb{R}^n$ is the unit vector in the j-th direction, i.e., the j-th column of an identity matrix with size n. The problems (4) and (5) are quadratically-constrained programming (QCP) optimization problems, which can be solved

with QCP solvers such as CPLEX. However, it is computationally expensive, since at least $2n$ QCPs have to be solved in each iteration.

This approach is essentially a QCP version of optimization-based bound tightening (OBBT) as used in the MINLP literature [23]. While typically performed with linear constraints, OBBT is often only done at the root node of the search tree as it is too expensive to perform at every node. Convex QCPs can be solved in polynomial time using iterative approaches such as the interior point method [24].

4.2   Axis-Aligned Tangent Box Filtering (BOX)

The simplest way to tighten the domains of the variables is to compute the tangent box of the hyper-ellipsoid defined in Eq. (2), where the edges of the box are parallel to the axes of the coordinate system. As a 2D example, the dotted box in Fig. 1 shows the tangent box. The lower bound $\boldsymbol{l}^b$ and the upper bound $\boldsymbol{u}^b$ that define the tangent box can be computed by solving the following problems:

$$l_j^b = \min_{\boldsymbol{x} \in \mathbb{R}^n} \boldsymbol{e}_j^T \boldsymbol{x} \quad \text{subject to } \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2 \leq \sqrt{\beta}, \tag{6}$$

$$u_j^b = \max_{\boldsymbol{x} \in \mathbb{R}^n} \boldsymbol{e}_j^T \boldsymbol{x} \quad \text{subject to } \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2 \leq \sqrt{\beta}. \tag{7}$$

Chang & Golub [3] proposed an efficient way to solve the above problems. We first solve the problem in (7) for u_j^b, and the lower bound l_j^b can be obtained by using the symmetric property of an ellipsoid. Let $\boldsymbol{p} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}$, the problem (7) becomes

$$
\begin{aligned}
u_j^b &= \max_{\boldsymbol{p}} \boldsymbol{e}_j^T \boldsymbol{A}^{-1}(\boldsymbol{p} + \boldsymbol{y}) \\
&= \max_{\boldsymbol{p}} \boldsymbol{e}_j^T \boldsymbol{A}^{-1}\boldsymbol{p} + \boldsymbol{e}_j^T \boldsymbol{A}^{-1}\boldsymbol{y} \quad \text{subject to } \|\boldsymbol{p}\|_2 \leq \sqrt{\beta}.
\end{aligned} \tag{8}
$$

By the Cauchy-Schwarz inequality, we have

$$\boldsymbol{e}_j^T \boldsymbol{A}^{-1}\boldsymbol{p} \leq \left\|\boldsymbol{A}^{-T}\boldsymbol{e}_j\right\|_2 \|\boldsymbol{p}\|_2 \leq \left\|\boldsymbol{A}^{-T}\boldsymbol{e}_j\right\| \sqrt{\beta}.$$

The first inequality becomes an equality if and only if $\boldsymbol{p} = c\boldsymbol{A}^{-T}\boldsymbol{e}_j$ for some non-negative scalar c. The second inequality becomes equality if and only if $\|\boldsymbol{p}\|_2 = \sqrt{\beta}$. Therefore, $\boldsymbol{p}$ is the minimizer for (8) when $\boldsymbol{p} = \sqrt{\beta}\boldsymbol{A}^{-T}\boldsymbol{e}_j / \left\|\boldsymbol{A}^{-T}\boldsymbol{e}_j\right\|_2$. Substituting $\boldsymbol{p}$ into (8), we have

$$u_j^b = \sqrt{\beta}\left\|\boldsymbol{A}^{-T}\boldsymbol{e}_j\right\|_2 + \boldsymbol{e}_j^T \boldsymbol{A}^{-1}\boldsymbol{y}. \tag{9}$$

From the symmetry property of an ellipsoid, we have

$$l_j^b = -\sqrt{\beta}\left\|\boldsymbol{A}^{-T}\boldsymbol{e}_j\right\|_2 + \boldsymbol{e}_j^T \boldsymbol{A}^{-1}\boldsymbol{y}. \tag{10}$$

Computing the Reduced Intersecting Ellipsoid $\mathcal{E}_{\mathcal{F}}$. When a variable is fixed during the search, e.g., $x_i = v_i$, the dimension of the ellipsoid is reduced by one. Geometrically, we need to find the one-dimension-smaller ellipsoid that intersects at $x_i = v_i$ and $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \le \beta$. We propose a general way to compute the reduced intersecting ellipsoid with any number of variables fixed.

Let $\mathcal{F}$ be the set of the variables that are fixed and let $\tilde{\boldsymbol{A}} = [\boldsymbol{A}_j], \forall j \ne \mathcal{F}$, $\bar{\boldsymbol{y}} = \boldsymbol{y} - \sum_{i \in \mathcal{F}} \boldsymbol{A}_i v_i$ and the QR factorization of $\tilde{\boldsymbol{A}}$ as: $\tilde{\boldsymbol{A}} = \begin{bmatrix} \tilde{\boldsymbol{Q}}_1 & \tilde{\boldsymbol{Q}}_2 \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{R}} \\ \boldsymbol{0} \end{bmatrix}$, the reduced intersecting ellipsoid $\mathcal{E}_{\mathcal{F}}$ can be computed as follows:

$$\mathcal{E}_{\mathcal{F}} = \left\| \tilde{\boldsymbol{y}} - \tilde{\boldsymbol{R}}\tilde{\boldsymbol{x}} \right\|_2^2 \le \tilde{\beta}, \tag{11}$$

where $\tilde{x}$ is the vector of the unknown variables of the reduced ellipsoid, $\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{Q}}_1^T \bar{\boldsymbol{y}}$ and $\tilde{\beta} = \beta - \|\bar{\boldsymbol{y}}\|_2^2 + \|\tilde{\boldsymbol{y}}\|_2^2$. The derivations on $\tilde{\boldsymbol{R}}$ and $\tilde{\boldsymbol{y}}$ are straightforward so we only explain $\tilde{\beta}$ as follows. We know that

$$\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 = \left\| \bar{\boldsymbol{y}} - \tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} \right\|_2^2 = \left\| \begin{bmatrix} \tilde{\boldsymbol{Q}}_1^T \\ \tilde{\boldsymbol{Q}}_2^T \end{bmatrix} \bar{\boldsymbol{y}} - \begin{bmatrix} \tilde{\boldsymbol{R}} \\ \boldsymbol{0} \end{bmatrix} \tilde{\boldsymbol{x}} \right\|_2^2.$$

It follows that

$$\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 - \left\| \tilde{\boldsymbol{Q}}_1^T \bar{\boldsymbol{y}} - \tilde{\boldsymbol{R}}\tilde{\boldsymbol{x}} \right\|_2^2 = \left\| \tilde{\boldsymbol{Q}}_2^T \bar{\boldsymbol{y}} \right\|_2^2 = \|\bar{\boldsymbol{y}}\|_2^2 - \left\| \tilde{\boldsymbol{Q}}_1^T \bar{\boldsymbol{y}} \right\|_2^2.$$

We assume that β and $\tilde{\beta}$ constraints are satisfied at equality as this corresponds to the largest ellipsoids defined by our inequalities and therefore ensures that no valid values are pruned. Since $\beta = \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$ and $\tilde{\beta} = \left\| \tilde{\boldsymbol{Q}}_1^T \bar{\boldsymbol{y}} - \tilde{\boldsymbol{R}}\tilde{\boldsymbol{x}} \right\|_2^2$, we have $\tilde{\beta} = \beta - \|\bar{\boldsymbol{y}}\|_2^2 + \|\tilde{\boldsymbol{y}}\|_2^2$.

At each node of the tree, we can first compute the reduced ellipsoid $\mathcal{E}_F$ w.r.t the variables that are already fixed with Eq. (11), then apply Eqs. (9) and (10) to compute the axis-aligned tangent box. The algorithm propagates when variables are instantiated or β is reduced. As our CP search instantiates variables, the propagation is active at each node.

Complexity of the Filtering Algorithm. The filtering algorithm reduces the domains of all the variables based on β with $O(n^3)$ time-complexity. First, computing the reduced ellipsoid takes $O(n^3)$, as the QR factorization is required, Second, computing the tangent box takes $O(n^3)$, as the complexity is dominated by computing the matrix inverse $\boldsymbol{A}^{-1}$. Therefore the total time complexity for filtering the domain for all the variable is $O(n^3)$.

4.3   Approximate Bounds Consistency (ABC) Filtering

Before we introduce the next filtering algorithm, our notation is summarized as follows:

- $[l_j, u_j]$: The interval domains (local bounds) of variable x_j.
- $[l_j^b, u_j^b]$: The tangent box derived with Eqs. (9) and (10) of the ellipsoid $\mathcal{E}$ defined in Eq. (2).
- v_j: A value that is within x_j's domain, i.e., $v_j \in I_D(x_j) = [l_j, u_j]$.

It is observed that if $\boldsymbol{l} \leq \boldsymbol{l}^b \leq \boldsymbol{u}^b \leq \boldsymbol{u}$, then the tangent box defines the bounds of the variables. However, a pair of bounds may lead to reductions in other variable domains. For example, in Fig. 2-a, the bounds l_i and u_i have the effect of increasing the lower bound l_j^b to l_j and decreasing the upper bound u_j^b to u_j.

To perform stronger domain reductions on the ellipsoid, first we need to determine the set of variables that can be used to infer reductions on the lower bounds or upper bounds of the variables. We explain the propagation algorithm below for the lower bound only since the propagation on the upper bound can be derived in a symmetric manner.

Proposition 1. *Let $P(\mathcal{E})_{ij}$ be the ellipse defined by the projection of the hyperellipsoid (Eq. 2) onto the $x_i x_j$ plane. Then the variable x_i can be used to infer domain reductions on x_j's lower bound if and only if $l_i \leq u_i \leq t_{ij}^l$ or $t_{ij}^l \leq l_i \leq u_i$, where t_{ij}^l is the x_i value at the intersection of $x_j = l_j^b$ and the projected ellipse $P(\mathcal{E})_{ij}$.*

Proof. If $l_i \leq t_{ij}^l \leq u_i$ (Fig. 2-b), we can set $x_i = t_{ij}^l$, so that $x_j = l_j^b$, thus no domain reduction can be inferred to x_j's lower bound. In the other two cases where $l_i \leq u_i \leq t_{ij}^l$ (Fig. 2-a) or $t_{ij}^l \leq l_i \leq u_i$, since x_j is forced to take a value that is greater than l_j^b, we can increase x_j's lower bound.

We refer to t_{ij}^l and t_{ij}^u as the touching points.

Computing the Touching Points. The touching points can be computed easily as a by-product of computing the axis-aligned tangent box (9) and (10).
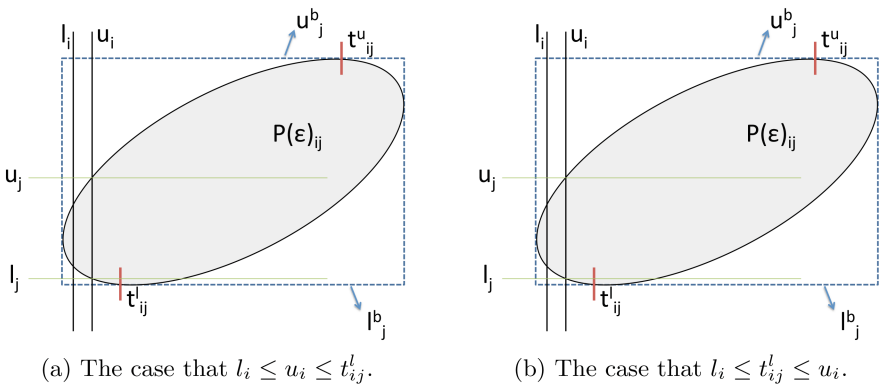

(a) The case that $l_i \leq u_i \leq t_{ij}^l$.   (b) The case that $l_i \leq t_{ij}^l \leq u_i$.

Fig. 2. The 2D projection of the hyper-ellipsoid onto the $x_i x_j$ plane.

Since $\boldsymbol{p} = \sqrt{\beta}\boldsymbol{A}^{-T}\boldsymbol{e}_j / \left\|\boldsymbol{A}^{-T}\boldsymbol{e}_j\right\|_2$ uniquely defines the minimizer for the upper bound u_j^b, let $\boldsymbol{x}^*$ be the solution to the equation $\boldsymbol{p} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}$, we have:

$$(\boldsymbol{t}_j^u)^T = [t_{1j}^u, \ldots, t_{j-1,j}^u, t_{j+1,j}^u, \ldots, t_{nj}^u]^T = [x_1^*, \ldots, x_{j-1}^*, x_{j+1}^*, \ldots, x_n^*]^T$$

Note that $\boldsymbol{t}_j^u$ is a $n-1$ dimensional vector and $x_j^* = u_j^b$. Using the symmetry property of the ellipsoid, $\boldsymbol{t}_j^l$ can be computed by reflecting t_j^u about the center of the ellipsoid.

The complexity of computing the touching points for all the variables, i.e., $\boldsymbol{t}_j^l, \boldsymbol{t}_j^u, \forall j$, is $O(n^3)$, as $\boldsymbol{x}^*$ can be computed in $O(n^2)$ for each variable, given that $\boldsymbol{A}^{-1}$ is known.

Proposition 2. *If x_i can be used to increase x_j's lower bound l_j according to Proposition 1, the value v_i^d that should be used to increase l_j is defined as*

$$v_i^d = \begin{cases} l_i, \text{ if } (t_{ij}^l - l_i)^2 \le (t_{ij}^l - u_i)^2. \\ u_i, \qquad \text{otherwise.} \end{cases}$$

Proof. Since $P(\mathcal{E})_{ij}$ is convex and x_j achieves its minimum l_j^b at $x_i = t_{ij}^l$, for any point x_j in $P(\mathcal{E})_{ij}$, we have x_j strictly larger than l_j^b if x_i takes any value other than t_{ij}^l. That is, x_j increases strictly when x_i moves away from t_{ij}^l. Therefore, the value (l_i or u_i) that achieves the minimum of the expression $\min((t_{ij}^l - u_i)^2, (t_{ij}^l - l_i)^2)$ determines x_j's lower bound.

As Fig. 2-a depicts, we choose u_i in this example, as using l_i removes the valid value l_j.

Using Proposition 1 and 2, we can identify the set of variables and their values that can be used to increase the lower bound of a variable.

Definition 2. *For each variable x_j, let S_j^l (S_j^u) be the set of all the variables that can be used to infer domain reductions on x_j's lower (upper) bound.*

Definition 3. *An assignment A: $x_i \mapsto I_D(x_i)$, $i \in S_j^l$ or S_j^u is said to be a determining assignment when $A(x_i) = v_i^d$.*

When a variable takes a determining assignment, we can compute the reduced intersecting ellipsoid using the method in Sect. 4.2. The complete filtering algorithm for pruning the lower bound of a variable is presented in Algorithm 1. The upper bound pruning can be derived in a symmetric manner.

Complexity of the Filtering Algorithm. The filtering algorithm reduces the domain of a single variable in $O(n^3)$ time-complexity. First, computing the tangent box takes $O(n^3)$ (see Sect. 4.2). The touching points require $O(n^2)$ as explained previously. In line seven, the sets S_j^l can be computed in $O(n)$. The for-loop at Line 8 requires $O(n^3)$, as Line 9 and 10 require $O(n^2)$ for computing the reduced ellipsoid (by updating the QR factorization) and the tangent plane for each of the i in S_j^l. Therefore the total time-complexity for filtering the domain for one variable is $O(n^3)$.

Algorithm 1. $Prune(l_j)$

1: **Data:** The local bounds: $l_1, \ldots, l_n, u_1, \ldots, u_n$, the tangent box for the ellipsoid $\mathcal{E}$: $l_1^b, \ldots, l_n^b, u_1^b, \ldots, u_n^b$, the touching points t_{ij}^l, β
2: **Results:** The filtered lower bound l_j'
3: **Initialization:** Set $l_j' = -\infty$, $\mathcal{F} = \{\}$
4: **if** $u_j < l_j^b$ **then**
5:    The constraint is not satisfiable
6: **else**
7:    Compute the set S_j^l and the associated assignments $A(x_i)$, $\forall i \in S_j^l$
8:    **for** each $i \in S_j^l$ **do**
9:       Let $\mathcal{F} = \{i\}$, compute the reduced intersecting ellipsoid $\mathcal{E}_F$
10:       Compute the tangent box $(l_j^F)^b$ of $\mathcal{E}_F$
11:       Set $l_j' = \max((l_j^F)^b, l_j')$
12:    **end for**
13:    **if** $u_i < l_j'$ **then**
14:       The constraint is not satisfiable
15:    **else**
16:       Set $l_j' = \max((l_j', l_j)$
17:    **end if**
18: **end if**

The complexity for filtering all the variables is therefore $O(n^4)$, which is the complexity when $S_j^l, \forall j$, contains $n-1$ variables. However, it is beneficial to have more variables in the set, as more and stronger pruning might be done.

4.4   Relative Strength of the Three Filtering Algorithms

It is clear that the ABC filtering algorithm is at least as strong as the BOX filtering algorithm, since ABC uses the tangent box as the starting point. The QCP filtering is at least as strong as ABC. ABC only considers the 2D projection of the hyper-ellipsoid onto each $x_i x_j$ plane, i.e., x_j is only tightened using the bounds of each x_i, independently. However, it is also possible to perform a higher dimension projection of the hyper-ellipsoid and use the bounds of more than one variable together to do bound tightening. Consider the 3D projection of the hyper-ellipsoid and the reasoning among x_i, x_j, x_k. It is possible to use x_i and x_k, together, to tighten x_j, given that x_i and x_k intersects inside the hyper-ellipsoid. For this reason, ABC only achieves BC when $|S_j^l| = 1, |S_j^u| = 1, \forall j$, and the tangent box only achieves BC when $|S_j^l| = 0, |S_j^u| = 0, \forall j$.

5   Branching Rules

We propose variable and value ordering rules inspired by the search strategy of DEBS. Recall that the static variable ordering in DEBS tries to minimize the branching factor at the top of the tree so that it is easier to find feasible solutions. In CP, this is the same as choosing a variable with minimum domain

size. We therefore use the standard dynamic variable selection rule that chooses a variable with the smallest domain. The value ordering rule in DEBS always assigns a variable to the integer value closest to the center of the ellipsoid of the objective function so that the search greedily chooses the best integer value at the current node with the hope of finding good feasible solution quickly. In our implementation, suppose x_j is the variable chosen for branching, we first compute the center of the ellipsoid c_j in j-th dimension given the reduced ellipsoid, and then round it to the nearest integer. When the search backtracks, we assign x_j to the next nearest integer to c_j, and so on.

6   Experimental Results

Experimental Setup. We design two experiments. The goal of the first experiment is to evaluate the impact of our filtering algorithms and the branching heuristics compared to a default CP model. The second experiment compares our new CP approach to the best known exact approaches for IQCPs.

For the first experiment, we use IBM ILOG CP Optimizer v12.6.3 with its default settings. The three filtering algorithms, e.g., BOX, ABC, and QCP, are implemented in CP Optimizer as customized global constraints. The branching rules are implemented as customized variable and value choosers (denoted with the symbol "+b" in our results). We use CPLEX v12.6.3 for solving the QCPs. We report the arithmetic mean CPU time "time" in seconds, and the arithmetic mean number of choice points "chpts" to find and prove optimality for each problem set.

For the second experiment, we use CPLEX v12.6.3,[2] BARON v16.4.7 (using CPLEX v12.6.3 as its LP/MIP solver) and the SDP solver BiqCrunch downloaded from the website [19] for comparison. All solvers are executed with their default settings.[3] The DEBS algorithm is written in C.

The CPU time limit for each run on each problem instance is 3600 s. All experiments were performed on a Intel(R) Xeon(R) CPU E5-1650 v2 3.50 GHz machine (in 64 bit mode) with 16 GB memory running MAC OS X 10.9.2 with one thread.

6.1   Problem Sets

We experiment on medium size problems in five problem classes. The problem size of each set is chosen with the aim for a mix of solvable and non-solvable instances across the default CP Optimizer and the three filtering algorithms.

Binary Quadratic Programming (BQP) Problem. The BQP problem is defined as: $\min_{x \in \{0,1\}} \frac{1}{2} x^T H x + f^T x$, where $H \in \mathbb{R}^{n \times n}$ and $f \in \mathbb{R}^n$. BQPs

---

[2] A major improvement was made in solving IQCPs in CPLEX v12.6.3 [25].

[3] There are four versions of the SDP solver that deal with problem-specific structures. The SDP results presented are the best version for each individual problem *instance*, representing the "virtual best" SDP solver.

arise in many combinatorial optimization problems such as task allocation [26], quadratic assignment [27], and max-cut problems [18]. We experiment on the Carter type problems [28] divided into four sub-sets of instances with size 40 ($p = 0.2$), 40 ($p = 0.3$), 50 ($p = 0.2$) and 50 ($p = 0.3$), respectively, where p is a problem generation parameter.

Exact Quadratic Knapsack Problem (EQKP). The EQKP [29] is defined as: $\min_{\boldsymbol{x} \in \mathcal{C}} \frac{1}{2} \boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} + \boldsymbol{f}^T \boldsymbol{x}$, $\mathcal{C} = \{\boldsymbol{x} \in \{0, 1\} : \boldsymbol{c}_1^T \boldsymbol{x} = K, \ \boldsymbol{c}_2^T \boldsymbol{x} \leq B\}$, where $\boldsymbol{H} \in \mathbb{R}^{n \times n}$, $\boldsymbol{f} \in \mathbb{R}^n$, $\boldsymbol{c}_1 \in \mathbb{R}^n$ is a vector equal to ones, $\boldsymbol{c}_2 \in \mathbb{R}^n_+$, $K \in \mathbb{Z}_+$, $B \in \mathbb{R}^+$. The objective is to minimize a quadratic function subject to a cardinality constraint and a knapsack constraint. The EQKP is a extension of the BQP, maximum diversity problem [30], quadratic knapsack problem [31], and exact linear knapsack problem [32]. EQKPs arise in a wide range of real world applications such as wind farm optimization [33,34]. We experiment on the EQKP and a variation from Ku & Beck [12] with binary domains $x_j \in \{0, 1\}$ relaxed to $x_j \in \{0, 1, 2\}$. We use three sub-sets of the instances with size 10, 20 and 30.

Box-constrained ILS (BILS) Problem. The BILS problem can be defined as: $\min_{\boldsymbol{x} \in \mathcal{C}} \|\boldsymbol{y} - \boldsymbol{A} \boldsymbol{x}\|_2^2$, $\mathcal{C} = \{\boldsymbol{x} \in \mathbb{Z}^n : \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \ \boldsymbol{l} \in \mathbb{Z}^n, \ \boldsymbol{u} \in \mathbb{Z}^n\}$. The problem minimizes a least squares expression subject to general integer bounds. Such problems exist in elevator scheduling [5] and signal processing [10]. We generate problems the same way as Chang et al. [10], with medium size variable domains ($0 \leq x_i \leq 10, \forall i$) and medium level of noise ($\sigma = 0.05$). We use five sub-sets of the instances with size 10, 20, 30, 40 and 50.

Box-constrained and Ellipsoid-constrained ILS (BEILS) Problem. The BEILS problem can be defined as: $\min_{\boldsymbol{x} \in \mathcal{C}} \|\boldsymbol{y} - \boldsymbol{A} \boldsymbol{x}\|_2^2$, $\mathcal{C} = \{\boldsymbol{x} \in \mathbb{Z}^n : \|\boldsymbol{A} \boldsymbol{x}\| \leq \alpha^2, \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \ \boldsymbol{l} \in \mathbb{Z}^n, \ \boldsymbol{u} \in \mathbb{Z}^n\}$, where α is a constant. In addition to the least squares objective function, the BEILS problem is also subject to a least squares constraint. Such problem can exist in signal processing [35]. We generate problems the same way as those in Chang et al. [3] on the ellipsoid-constrained ILS problem then add medium size variable domains ($-10 \leq x_i \leq 10, \forall i$). We use five sub-sets of the instances with size 10, 20, 30, 40 and 50.

Quadratic Lateness Scheduling Problem (QLSP). The QLSP can be defined as: $\min \sum_{j=1}^{n} (S_j + p_j - d_j)^2$ s.t. `disjunctive`($\{S_1, ..., S_n\}$, $\{p_1, ..., p_n\}$), $S_j \geq 0, \forall j$, where S_j, p_j, and d_j are the start time, processing time, and due date of job j. The QLSP is a single machine scheduling problem with the goal of minimizing the sum of the quadratic lateness of the jobs. We generate problems the same way as those in Schaller [36]. We use four sub-sets of the instances with size 5, 10, 15 and 20.

Each problem set includes 10 instances of each size, e.g., the BQP problems includes 4 sub-sets of size 10 for 40 instances.

6.2 Results of Experiment 1

We present the results of Experiment 1 in Tables 1 and 2. From Table 1, it is clear that the ELLIPSOID constraint significantly improves the performance of

the default CP Optimizer for the BQPs, the BILS problems, and the BEILS problems both in terms of running time and number of choice points. Without the reasoning from the ELLIPSOID constraint, the default CP Optimizer cannot prove optimality for any instances of these three problem types. For the EQKPs, the ELLIPSOID constraint (BOX) is able to decrease the number of choice points by a factor of 1.5 for the {0,1} problems and almost a factor of 2 for the {0,1,2} problems. But the extra computation makes the running time worse than that of the default CP Optimizer.

For QLSPs, CPO achieves the best average running time. Interestingly, for the instances where all the filtering algorithms are able to prove optimality, the number of choice points is the same for all the filtering algorithms. Our preliminary investigation shows that the `disjunctive` constraint has a significant impact on reducing variable domains for the QLSP, therefore determining the number of choice points regardless of the strength of propagation of the ellipsoid constraint. It is worth pointing out that the convex ellipsoid structure of the QLSP has a simple *axis-aligned* structure, i.e., the off-diagonal entries of H in Eq. (3) are all equal to zero. Our results show that the default CPO propagation does not achieve BC for axis-aligned ellipsoids and that the strength of pruning of our three inference algorithm follows the same pattern for axis-aligned ellipsoids as for the general case. As future work, we would like to exploit the axis-aligned property to achieve more efficient filtering.

Among the three filtering algorithms, BOX performs the best in terms of running time to prove optimality. ABC and QCP both find better primal solutions in fewer nodes but, BOX finds better solutions in less time. On problems where all three algorithms are able to prove optimality, the tree size of BOX is about 15 % larger than that of ABC and QCP. This is somewhat surprising and suggests that variable fixing leads to strong inference. We observe that the reduced ellipsoid obtained after fixing a variable often has a much tighter tangent box on the unfixed variables compared to that of the original ellipsoid. However, we also observe that ABC can sometimes achieve one order of magnitude improvement compared to BOX in tree size on some larger instances. We would like to investigate the problem characteristics that result in such difference. Note that the lower number of choice points of ABC and QCP are misleading because neither prove optimality for all instances within the time limit. However they are good indicators on the number of nodes that can be visited when these two algorithms are applied.

From Table 2, it is observed that the new branching rules greatly improve the number of choice points, the running time, and the percentage of optimal solutions found for most approaches. However CPO still cannot prove optimality for the BQPs, the BILS problems, and the BEILS problems. The most significant reduction is observed on the BILS problem and the BEILS problem, followed by the QLSP, where the variable domains are much larger than the other two types of problems: a good branching strategy apparently is especially important for problems with large domains. It is particularly striking to see that the BILS problems are solved at least five orders of magnitude faster by CP through the use of the ellipsoid constraint and the branching rule.

Table 1. A comparison of default CP Optimizer and the three filtering algorithms. Bold numbers indicate the best approach for a given problem set. The symbol '-' means that no problem instances were solved to optimality within 3600 s. The superscripts indicate the percentage of instances solved to optimality within 3600 s. If no superscript is indicated, all of the instances are solved.

Problem	CPO		BOX		ABC		QCP	
	time	chpts	time	chpts	time	chpts	time	chpts
BQP	-	-	**11.12**	**17169**	1204.89^{85}	10565	-	-
BILS	-	-	**44.61**	**61552**	871.80^{84}	19229	2810.01^{40}	3836
BEILS	-	-	**362.14^{94}**	**167162**	1480.68^{64}	10301	3092.14^{38}	1637
EQKP{0,1}	**23.48**	667740	44.09	**426255**	745.79^{87}	60789	2494.77^{67}	2494
EQKP{0,1,2}	**83.98**	1803982	99.70	**932035**	481.44^{90}	63177	2256.95^{57}	2273
QLSP	**136.34**	**962400**	166.16	**962400**	754.07^{85}	438394	2432.98^{50}	8631

Table 2. A comparison of default CP Optimizer and the three filtering algorithms with the branching rules. All notations are the same as in Table 1.

Problem	CPO+b		BOX+b		ABC+b		QCP+b	
	time	chpts	time	chpts	time	chpts	time	chpts
BQP	-	-	**8.28**	**12611**	1099.82^{92}	12575	3595.85^{3}	1069
BILS	-	-	**0.06**	225	5.21	228	314.16	**221**
BEILS	-	-	**0.47**	426	213.47	**394**	1713.89^{82}	360
EQKP{0,1}	**16.40**	247896	24.84	**193269**	578.74^{90}	72134	1868.71^{57}	2550
EQKP{0,1,2}	50.04	521037	**46.77**	**269937**	407.59^{90}	39076	2304.01^{60}	2365
QLSP	**20.63**	**347665**	31.47	**347665**	602.77^{90}	265252	2249.57^{50}	8516

6.3 Results of Experiment 2

In this section, we compare our best CP results (using the BOX filtering with the branching rules) with the best known exact approaches. From Table 3, it is observed that our new CP approach significantly outperforms the general MINLP solver BARON and it is competitive with state-of-the-art MIP solver CPLEX, running at the same order of magnitude as CPLEX for BQPs and BILS problems. While our CP approach is one order of magnitude slower than CPLEX for EQKPs, it is almost two orders of magnitude faster for BEILS problems and it is significantly better for QLSPs. The reason that CPLEX performs particularly poorly on QLSPs is that the modeling of the disjunctive relationship among the jobs involve big-M constraints, which result in weak dual bounds. As future work, we would like to apply our CP approach to even more complicated scheduling problems, where quadratic component is only one of many components.

The SDP approach, while being the state-of-the-art for the BQPs, is limited to problems with binary domains. Similarly, DEBS cannot be applied to all the

Table 3. A comparison of default CP Optimizer and our best setting: BOX+b. All notations are the same as in Table 1. The additional symbol "N/A" indicates that the problem cannot be solved with the approach.

Problem	CPO	BOX+b	CPLEX	BARON	DEBS	SDP
	time	time	time	time	time	time
BQP	-	8.28	37.06	38.03	**0.24**	0.69
BILS	-	0.06	0.02	2715.04^{26}	**0.01**	N/A
BEILS	-	**0.47**	14.79	3248.25^{16}	N/A	N/A
EQKP{0,1}	23.48	24.84	4.49	6.23	**0.24**	114.01
EQKP{0,1,2}	83.98	46.77	4.27	429.62^{93}	**0.34**	N/A
QLSP	136.34	**31.47**	1588.15^{65}	1855.02^{50}	N/A	N/A

problem classes due to its specialized nature. In contrast, BARON is the most general solver tested here (along with CPO) and these results on strictly convex IQCPs do not reflect its more general problem solving power [15].

7   Conclusion

We propose a CP-based approach to solve strictly convex IQCPs via a novel ELLIPSOID constraint with three different filtering algorithms and variable/value ordering heuristics. The constraint and branching heuristics are based on the geometry of the strictly convex quadratic function. We experiment with a variety of problems and show significant improvement over the default CP Optimizer and competitive results to general state-of-the-art solvers CPLEX and BARON.

For future work, it is interesting to experiment with our filtering algorithms on other problem types. Although ABC and QCP perform worse than BOX for the problem types tested here, it is possible that ABC and QCP can perform better for problems where variable fixings do not take place frequently. For the same reason it is also interesting to implement the filtering algorithms in a MIP-based solver such as SCIP [37] where branching is typically done by tightening variable bounds instead of fixing variables. In general, our technique can be integrated with other solvers provided they represent bounds on variables and have an "inference loop".

More broadly, we propose that it may be possible to develop CP as a basis for MINLP. MINLPs are challenging optimization problems that arise in many industrial applications. As CP is not dependent on a strong linear relaxation to bound the search, it may be valuable to study inferences that can be made for common non-linear constraints as we have done here. We hope that this paper might serve as a step in this direction.

References

1. van Emde-Boas, P.: Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Mathematisch Instituut, Amsterdam, The Netherlands (1981)
2. Agrell, E., Eriksson, T., Vardy, A., Zeger, K.: Closest point search in lattices. IEEE Trans. Inf. Theory **48**(8), 2201–2214 (2002)
3. Chang, X.W., Golub, G.H.: Solving ellipsoid-constrained integer least squares problems. SIAM J. Matrix Anal. Appl. **31**(3), 1071–1089 (2009)
4. Teunissen, P.J., Kleusberg, A., Teunissen, P.: GPS for Geodesy, vol. 2. Springer, Berlin (1998)
5. Kuusinen, J.M., Sorsa, J., Siikonen, M.L.: The elevator trip origin-destination matrix estimation problem. Transp. Sci. **49**(3), 559–576 (2014)
6. Pesant, G., Régin, J.-C.: SPREAD: a balancing constraint based on statistics. In: Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 460–474. Springer, Heidelberg (2005)
7. Domes, F., Neumaier, A.: Constraint propagation on quadratic constraints. Constraints **15**(3), 404–429 (2010)
8. Lebbah, Y., Michel, C., Rueher, M.: A rigorous global filtering algorithm for quadratic constraints. Constraints **10**(1), 47–65 (2005)
9. Ku, W.-Y., Beck, J.C.: Combining discrete ellipsoid-based search and branch-and-cut for binary quadratic programming problems. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 334–350. Springer, Heidelberg (2014)
10. Chang, X.W., Han, Q.: Solving box-constrained integer least squares problems. IEEE Trans. Wirel. Commun. **7**(1), 277–287 (2008)
11. Ku, W.Y., Beck, J.C.: Combining discrete ellipsoid-based search and branch-and-cut for integer least squares problems. Submitted to IEEE Trans. Wirel. Commun. (2014)
12. Ku, W.-Y., Beck, J.C.: Combining constraint propagation and discrete ellipsoid-based search to solve the exact quadratic knapsack problem. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 231–239. Springer, Heidelberg (2015)
13. Golub, G.H., Van Loan, C.F.: Matrix Computations, vol. 3. JHU Press, Baltimore (2012)
14. Sahinidis, N.V.: BARON 14.3.1: Global Optimization of Mixed-Integer Nonlinear Programs, User's Manual (2014)
15. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**, 225–249 (2005)
16. Misener, R., Floudas, C.A.: ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization of Nonlinear Equations. J. Global Optim. (2014). doi:10.1007/s10898-014-0166-2
17. Bussieck, M.R., Vigerske, S.: MINLP Solver Software. Wiley Encyclopedia of Operations Research and Management Science. Wiley, Chichester (2010)
18. Krislock, N., Malick, J., Roupin, F.: Improved semidefinite bounding procedure for solving max-cut problems to optimality. Math. Program. **143**(1), 61–86 (2012)
19. Krislock, N., Malick, J., Roupin, F.: BiqCrunch solver. http://lipn.univ-paris13.fr/BiqCrunch/download. Accessed 4 Dec 2016
20. Borno, M.A.: Reduction in solving some integer least squares problems. arXiv preprint arXiv:1101.0382 (2011)
21. Schnorr, C.P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Program. **66**(1), 181–199 (1994)
22. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)

23. Gleixner, A.M.: Exact and fast algorithms for mixed-integer nonlinear programming. Ph.D. thesis, Technische Universität Berlin (2015)
24. Nesterov, Y., Nemirovskii, A., Ye, Y.: Interior-point Polynomial Algorithms in Convex Programming, vol. 13. SIAM, Philadelphia (1994)
25. Bonami, P., Tramontani, A.: Advances in CPLEX for mixed integer nonlinear optimization. Presented at ISMp 2015, Pittsburgh, PA (2015)
26. Lewis, M., Alidaee, B., Kochenberger, G.: Using xqx to model and solve the uncapacitated task allocation problem. Oper. Res. Lett. **33**(2), 176–182 (2005)
27. FlNKE, G., Burkard, R.E., Rendl, F.: Quadratic assignment problems. Surv. Comb. Optim. **132**, 61–82 (2011)
28. Carter, M.W.: The indefinite zero-one quadratic problem. Discrete Appl. Math. **7**(1), 23–44 (1984)
29. Létocart, L., Plateau, M.C., Plateau, G.: An efficient hybrid heuristic method for the 0–1 exact k-item quadratic knapsack problem. Pesquisa Operacional **34**(1), 49–72 (2014)
30. Martí, R., Gallego, M., Duarte, A.: A branch and bound algorithm for the maximum diversity problem. Eur. J. Oper. Res. **200**(1), 36–44 (2010)
31. Caprara, A., Pisinger, D., Toth, P.: Exact solution of the quadratic knapsack problem. INFORMS J. Comput. **11**(2), 125–137 (1999)
32. Caprara, A., Kellerer, H., Pferschy, U., Pisinger, D.: Approximation algorithms for knapsack problems with cardinality constraints. Eur. J. Oper. Res. **123**(2), 333–345 (2000)
33. Turner, S., Romero, D., Zhang, P., Amon, C., Chan, T.: A new mathematical programming approach to optimize wind farm layouts. Renewable Energy **63**, 674–680 (2014)
34. Zhang, P.Y., Romero, D.A., Beck, J.C., Amon, C.H.: Solving wind farm layout optimization with mixed integer programs and constraint programs. EURO J. Comput. Optim. **2**(3), 195–219 (2014)
35. Damen, M.O., El Gamal, H., Caire, G.: On maximum-likelihood detection and the search for the closest lattice point. IEEE Trans. Inf. Theory **49**(10), 2389–2402 (2003)
36. Schaller, J.: Single machine scheduling with early and quadratic tardy penalties. Comput. Ind. Eng. **46**(3), 511–532 (2004)
37. Achterberg, T.: SCIP: solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009)

A Global Constraint for Closed Frequent Pattern Mining

Nadjib Lazaar[1(✉)], Yahia Lebbah[2], Samir Loudni[3], Mehdi Maamar[1,2],
Valentin Lemière[3], Christian Bessiere[1], and Patrice Boizumault[3]

[1] LIRMM, University of Montpellier, Montpellier, France
`lazaar@lirmm.fr`
[2] LITIO, University of Oran 1 Ahmed Ben Bella, Oran, Algeria
[3] GREYC, Normandie University, Caen, France

Abstract. Discovering the set of closed frequent patterns is one of the fundamental problems in Data Mining. Recent Constraint Programming (CP) approaches for declarative itemset mining have proven their usefulness and flexibility. But the wide use of reified constraints in current CP approaches leads to difficulties in coping with high dimensional datasets. In this paper, we propose the CLOSEDPATTERN global constraint to capture the closed frequent pattern mining problem without requiring reified constraints or extra variables. We present an algorithm to enforce *domain consistency* on CLOSEDPATTERN in polynomial time. The computational properties of this algorithm are analyzed and its practical effectiveness is experimentally evaluated.

1 Introduction

Frequent Pattern Mining is a well-known and perhaps the most popular research field of data mining. Originally introduced by Agrawal et al. [1], it plays a key role in many data mining applications. These applications include the discovery of frequent itemsets and association rules [1], correlations [2] and many other data mining tasks.

In practice, the number of frequent patterns produced is often huge and can easily exceed the size of the input dataset. Most frequent patterns are redundant and can be derived from other found patterns. Hence, *closed frequent patterns* have been introduced. They provide a concise and condensed representation that avoids redundancy. Discovering the set of closed frequent patterns is one of the fundamental problems in Data Mining. Several specialized approaches have been proposed to discover closed frequent patterns (e.g., A-CLOSE Algorithm [12], CHARM [17], CLOSET [13], LCM [14]).

Over the last decade, the use of the Constraint Programming paradigm (CP) to model and to solve Data Mining problems has received considerable attention [3,5,9]. The declarative aspect represents the key success of the proposed CP approaches. Doing so, one can add/remove any user-constraint without the need of developing specialized solving methods.

Related to the Closed Frequent Pattern Mining problem (CFPM), Guns et. al., propose to express the different constraints that we can have in Pattern Mining as a CP model [5]. The model is expressed on Boolean variables representing items and transactions, with a set of reified sum constraints. The reified model has become a de facto standard for many DM tasks. Indeed, the reified model had been adopted for the k-pattern sets [5]. The drawback is the wide use of reified constraints in the CP model, which makes the scalability of the approach questionable.

In the line of the work of Kemmar et al. [8], we propose in this paper the CLOSEDPATTERN global constraint. CLOSEDPATTERN does not require reified constraints and extra variables to encode and propagate the CFPM problem. CLOSEDPATTERN captures the particular semantics of the CFPM problem and domain consistency can be achieved on it using a polynomial algorithm. Experiments on several known large datasets show that our approach outperforms the reified model used in CP4IM [3] and is more scalable, which is a major issue for CP approaches. This result can be explained by the fact that CLOSEDPATTERN insures domain consistency.

The paper is organized as follows. Section 2 recalls preliminaries. Section 3 provides the context and the motivations for the CLOSEDPATTERN global constraint. Section 4 presents the global constraint CLOSEDPATTERN. Section 5 illustrates the power of the pruning algorithm compared with the reified model. Section 6 reports experiments. Finally, we conclude and draw some perspectives.

2   Background

In this section, we introduce some useful notions used in closed frequent pattern mining and constraint programming.

2.1   Closed Frequent Pattern Mining

Let $\mathcal{I} = \{1, ..., n\}$ be a set of n *item* indices[1] and $\mathcal{T} = \{1, ..., m\}$ a set of m *transaction* indices. A Pattern P (i.e., itemset) is a subset of $\mathcal{I}$. The language of patterns corresponds to $\mathcal{L}_{\mathcal{I}} = 2^{\mathcal{I}}$. A transaction database is a set $\mathcal{D} \subseteq \mathcal{I} \times \mathcal{T}$. The set of items corresponding to a transaction identified by t is denoted by $\mathcal{D}[t] = \{i \mid (i, t) \in \mathcal{D}\}$. A transaction t is an occurrence of some pattern P iff the set $\mathcal{D}[t]$ contains P (i.e., $P \subseteq \mathcal{D}[t]$).

The cover of P, denoted by $\mathcal{T}_{\mathcal{D}}(P)$, is the set of transactions containing P, that is, $\mathcal{T}_{\mathcal{D}}(P) = \{t \in \mathcal{T} \mid P \subseteq \mathcal{D}[t]\}$. Given $S \subseteq \mathcal{T}$ a subset of transactions, $\mathcal{I}_{\mathcal{D}}(S)$ is the set of common items of S, that is, $\mathcal{I}_{\mathcal{D}}(S) = \bigcap_{t \in S} \mathcal{D}[t]$. The (absolute) frequency of a pattern P is the size of its cover (i.e., $freq_{\mathcal{D}}(P) = |\mathcal{T}_{\mathcal{D}}(P)|$). Let $\theta \in \mathbb{N}^+$ be some given constant called a *minimum support*. A pattern P is frequent if $freq_{\mathcal{D}}(P) \geq \theta$.

---

[1] For the sake of readability, our examples refer to items by their names instead of their indices.

Example 1. Consider the transaction database in Table 1. We have $\mathcal{T}_{\mathcal{D}}(C) = \{t_1, t_3\}$, $freq_{\mathcal{D}}(C) = 2$ and $\mathcal{I}_{\mathcal{D}}(\{t_1, t_3\}) = CH$.

The closure of a pattern P in $\mathcal{D}$, denoted by $Clos(P)$, is the set of common items of its cover $\mathcal{T}_{\mathcal{D}}(P)$, that is, $Clos(P) = \mathcal{I}_{\mathcal{D}}(\mathcal{T}_{\mathcal{D}}(P))$. A pattern is closed if and only if $Clos(P) = P$.

Table 1. A transaction database $\mathcal{D}$ (a) and its binary matrix (b).

Trans.	Items					Trans.	A	B	C	D	E	F	G	H
t_1		B	C		G H	t_1	0	1	1	0	0	0	1	1
t_2	A		D			t_2	1	0	0	1	0	0	0	0
t_3	A		C D		H	t_3	1	0	1	1	0	0	0	1
t_4	A		E F			t_4	1	0	0	0	1	1	0	0
t_5	B		E F G			t_5	0	1	0	0	1	1	1	0

Definition 1 (Closed Frequent Pattern Mining (CFPM)). *Given a transaction database $\mathcal{D}$ and a minimum support threshold θ, the closed frequent pattern mining problem is the problem of finding all patterns P such that $(freq_{\mathcal{D}}(P) \geq \theta)$ and $(Clos(P) = P)$.*

Example 2. For $\theta = 2$, the set of closed frequent patterns in Table 1 is $\emptyset\langle 5\rangle$,[2] $A\langle 3\rangle$, $AD\langle 2\rangle$, $BG\langle 2\rangle$, $CH\langle 2\rangle$ and $EF\langle 2\rangle$.

Closed frequent patterns provide a minimal representation of frequent patterns, i.e. we can derive all frequent patterns with their exact frequency value from the closed ones [12]. We now define the important notion of *full extension* that comes from pattern mining algorithms and that we will use later in this paper.

Definition 2 (Full extension). *The non-empty itemset Q is called a full extension of P iff $\mathcal{T}_{\mathcal{D}}(P) = \mathcal{T}_{\mathcal{D}}(P \cup Q)$.*

Definition 2 is at the key of the item merging property [15] stated as follows: If some pattern Q is a full extension of some pattern P, and none of the proper supersets of Q is a full extension of P, then $P \cup Q$ forms a closed pattern. In other words, a closed pattern can be defined as a pattern that does not possess a full extension.

Search Space Issues. In pattern mining, the search space contains $2^{\mathcal{I}}$ candidates. Given a large number of items $\mathcal{I}$, a naive search that consists of enumerating and testing the frequency of pattern candidates in a dataset is infeasible. The main property exploited by most algorithms to reduce the search space is that frequency is *monotone decreasing* with respect to extension of a set.

Property 1 (Anti-monotonicity of the frequency). Given a transaction database $\mathcal{D}$ over $\mathcal{I}$, and two patterns $X, Y \subseteq \mathcal{I}$. Then, $X \subseteq Y \Rightarrow freq_{\mathcal{D}}(Y) \leq freq_{\mathcal{D}}(X)$.

Hence, any subset (resp. superset) of a frequent (resp. infrequent) pattern is also a frequent (resp. infrequent) pattern.

---

[2] Value between $\langle . \rangle$ indicates the frequency of a pattern.

2.2 CFPM Under Constraints

Constraint-based pattern mining aims at extracting all patterns P of $\mathcal{L}_\mathcal{I}$ satisfying a query $q(P)$ (conjunction of constraints), which usually defines what we call a *theory* [10]: $Th(q) = \{P \in \mathcal{L}_\mathcal{I} \mid q(P) \ is \ true\}$. A common example is the frequency measure leading to the frequent pattern constraint. It is also possible to have other kind of (user-)constraints. For instance, constraints on the size of the returned patterns, $minSize(P, \ell_{min})$ constraint holds if and only if the number of items of P is greater than or equal to ℓ_{min}. Constraints on the presence of an item in a pattern $item(P, i)$ state that an item i must be in a pattern P.

2.3 CSP and Global Constraints

A constraint network is defined by a set of variables $X = \{x_1, \ldots, x_n\}$, each variable $x_i \in X$ having an associated finite domain $dom(x_i)$ of possible values, and a set of constraints $\mathcal{C}$ on X. A constraint $c \in \mathcal{C}$ is a relation that specifies the allowed combinations of values for its variables $X(c)$. An assignment σ is a mapping from variables in X to values in their domains. The *Constraint Satisfaction Problem* (CSP) consists in finding an assignment satisfying all constraints.

Domain Consistency (DC). Constraint solvers typically use backtracking search to explore the search space of partial assignments. At each assignment, filtering algorithms prune the search space by enforcing local consistency properties like domain consistency. A constraint c on $X(c)$ is domain consistent, if and only if, for every $x_i \in X(c)$ and every $d_i \in dom(x_i)$, there is an assignment σ satisfying c such that $x_i = d_i$.

Global Constraints are constraints capturing a relation between a non-fixed number of variables. These constraints provide the solver with a better view of the structure of the problem. Examples of global constraints are AllDifferent, Regular and Among (see [7]). Except the case when for a given global constraint a Berge-acyclic decomposition exists, global constraints cannot be efficiently propagated by generic local consistency algorithms, which are exponential in the number of the variables of the constraint. Dedicated filtering algorithms are constructed to achieve polynomial time complexity in the size of the input, i.e., the domains and extra parameters. The aim of this paper is to propose a filtering algorithm for the frequent closed pattern constraint.

3 Context and Motivations

This section provides a critical review of ad-hoc specialized methods and CP approaches for CFPM, and motivates the proposition of a global constraint.

Specialized Methods for CFPM. CLOSE [12] was the first algorithm proposed to extract closed frequent patterns (CFPs). It uses an apriori-like bottom-up method. Later, Zaki and Hsiao [17] proposed a depth-first algorithm based

on a vertical database format e.g. `CHARM`. In [13], Pei et al. extended the FP-growth method to a method called `CLOSET` for mining CFPs. Lastly, Uno et al. [14] have proposed `LCM`, one of the fastest frequent itemset mining algorithm. It uses a hybrid representation based on vertical and horizontal representations. The milestone of LCM is a technique called *prefix preserving closure extension* (PPCE), which allows to generate a new frequent closed pattern from a previously obtained closed pattern. Let us explain the PPCE principle. Consider the closed pattern P. Let $P(i) = P \cap \{1, ..., i\}$ be the subset of P consisting of items no greater than i. The core index of P, denoted by $core(P)$, is the minimum index i such that $\mathcal{T}_{\mathcal{D}}(P(i)) = \mathcal{T}_{\mathcal{D}}(P)$. A pattern Q is PPCE of P if $Q = Clos(P \cup \{i\})$ and $P(i-1) = Q(i-1)$ for an item $i \notin P$ and $i > core(P)$. The completeness of PPCE is guaranteed by the following property: If Q is a nonempty closed itemset, then there is only one closed itemset P such that Q is a PPCE of P. The mining process using `LCM` is a depth first search where at each node we have a closed pattern P. `LCM` uses the PPCE technique as a branching strategy to jump from a closed pattern to other closed patterns by adding new items. With its specialized depth first search, `LCM` succeeds to enumerate very quickly the closed patterns. However, if the user considers other (user-)constraints on patterns, the search procedure should be revised. In fact, all these specialized proposals (e.g., Closet, Charm, `LCM`, etc.), though efficient, are ad-hoc methods suffering from the lack of genericity, since adding new constraints requires new implementations.

Reified Constraint Model for Itemset Mining. De Raedt et al. have proposed in [3] a CP model for itemset mining. They show how some constraints (e.g., frequency, maximality, closedness) can be modeled as CSP [6,11]. This modeling uses two sets of Boolean variables P and T: (1) Decision variables: item variables $P_1, P_2, ..., P_n$, where $P_i = 1$ if and only if item i is in the searched pattern; (2) Auxiliary variables: transaction variables $T_1, T_2, ..., T_m$, where $T_t = 1$ if and only if the searched pattern is in $\mathcal{D}[t]$.

The relationship between P and T, set of channeling constraints, is modeled by reified constraints stating that, for each transaction t, $(T_t = 1)$ iff P is a subset of $\mathcal{D}[t]$: $\forall t \in \mathcal{T} : (T_t = 1) \leftrightarrow \sum_{i \in \mathcal{I}} P_i(1 - \mathcal{D}[t,i]) = 0$ (arity $n + 1$). The min frequency constraint is modeled us: $\forall i \in \mathcal{I} : (P_i = 1) \rightarrow \sum_{t \in T} T_t \mathcal{D}[t,i] \geq \theta$ (arity $m + 1$). The closedness constraint is expressed with: $\forall i \in \mathcal{I} : (P_i = 1) \leftrightarrow \sum_{t \in T} T_t(1 - \mathcal{D}[t,i]) = 0$ (arity $m + 1$). Such encoding has a major drawback since it requires $(m + n + n)$ reified constraints of arity $(n + 1)$ and $(m + 1)$ to encode the whole database. This constitutes a strong limitation especially when it comes to handle very large databases.

We propose in the next section the CLOSEDPATTERN global constraint to encode both the minimum frequency constraint and the closedness constraint. This global constraint requires neither reified constraints nor auxiliary variables.

4   CLOSEDPATTERN Constraint

This section presents the CLOSEDPATTERN global constraint for the CFPM problem.

4.1   Definition and Filtering

Let P be the unknown pattern we are looking for. The unknown pattern P is encoded with Boolean item variables $P_1, ..., P_n$. In the rest of the paper we will denote by σ the partial assignment obtained from the variables $P_1, ..., P_n$ that have a singleton domain. We will also use the following subsets of items:

– present items: $\sigma^+ = \{j \in 1..n \mid P_j = 1\}$,
– absent items: $\sigma^- = \{j \in 1..n \mid P_j = 0\}$,
– other items: $\sigma^* = \{1..n\} \setminus (\sigma^+ \cup \sigma^-)$.

σ^* is the set of free items (non instantiated variables). If $\sigma^* = \emptyset$ then σ is a complete assignment.

The global constraint CLOSEDPATTERN ensures both the minimum frequency property and the closedness property.

Definition 3 (CLOSEDPATTERN global constraint). *Let $P_1, \ldots, P_n$ be binary item variables. Let $\mathcal{D}$ be a transaction database and θ a minimum support. Given a complete assignment σ on $P_1, \ldots, P_n$, CLOSEDPATTERN $_{\mathcal{D},\theta}(\sigma)$ holds if and only if $freq_{\mathcal{D}}(\sigma^+) \geq \theta$ and σ^+ is closed.*

Example 3. Consider the transaction database of Table 1a with $\theta = 2$. Let $P = \langle P_1, \ldots, P_8 \rangle$ with $dom(P_i) = \{0, 1\}$ for $i \in 1..8$. Consider the closed pattern AD encoded by $P = \langle 10010000 \rangle$, where $\sigma^+ = \{A, D\}$ and $\sigma^- = \{B, C, E, F, G, H\}$. CLOSEDPATTERN$_{\mathcal{D},2}(P)$ holds because $freq_{\mathcal{D}}(\{A, D\}) \geq 2$ and $\{A, D\}$ is closed.

Let σ be a partial assignment of variables P and i a free item. We use the vertical representation of the dataset, denoted $\mathcal{V}_{\mathcal{D}}$ where for each item, the transactions containing it are stored: $\forall i \in \mathcal{I}, \mathcal{V}_{\mathcal{D}}(i) = \mathcal{T}_{\mathcal{D}}(\{i\})$. We denote by $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i)$ the cover of item i within the current cover of a pattern σ^+:

$$\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i) = \mathcal{T}_{\mathcal{D}}(\sigma^+ \cup \{i\}) = \mathcal{T}_{\mathcal{D}}(\sigma^+) \cap \mathcal{T}_{\mathcal{D}}(\{i\}).$$

We need to define extensible assignments.

Definition 4 (Extensible assignment). *Given a constraint CLOSEDPATTERN $_{\mathcal{D},\theta}$ on $P_1, \ldots, P_n$, a partial assignment is said to be extensible if and only if it can be extended to a complete assignment of $P_1, \ldots, P_n$ that satisfies CLOSEDPATTERN $_{\mathcal{D},\theta}$.*

We show when a partial assignment is extensible with respect to CLOSEDPATTERN constraint.

Proposition 1. *Let σ be a partial assignment of variables in $P_1, \ldots, P_n$. σ is an extensible partial assignment if and only if $freq_{\mathcal{D}}(\sigma^+) \geq \theta$ and $\nexists j \in \sigma^-$ such that $\{j\}$ is a full extension of σ.*

Proof. According to the anti-monotonicity property of the frequency (cf. Property 1), if the partial assignment σ is infrequent (i.e., $freq_{\mathcal{D}}(\sigma^+) < \theta$), it cannot, under any circumstances, be extended to a closed pattern.

Given now a frequent partial assignment σ (i.e., $freq_{\mathcal{D}}(\sigma^+) \geq \theta$), let us take $j \in \sigma^-$ such that $\{j\}$ is a full extension of σ. It follows that $\mathcal{T}_{\mathcal{D}}(\sigma^+) = \mathcal{T}_{\mathcal{D}}(\sigma^+ \cup \{j\}) = \mathcal{V}_{\mathcal{D}}^{\sigma^+}(j)$. Therefore, $Clos(\sigma^+) = Clos(\sigma^+ \cup \{j\})$. Since σ^+ without j (j being in σ^-) cannot be extended to a closed pattern, the result follows. If there is no item $j \in \sigma^-$ such that $\{j\}$ is a full extension of σ, then the current assignment σ can be definitely extended to a closed itemset by adopting a full extension to form a closed pattern. □

We now give the CLOSEDPATTERN filtering rules by showing when a value of a given variable is inconsistent.

Proposition 2 (CLOSEDPATTERN filtering rules). *Let σ be an extensible partial assignment of variables in $P_1, \ldots, P_n$, and P_j ($j \in \sigma^*$) be a free variable. The following two cases characterize the inconsistency of the values 0 and 1 of P_j:*

- *$0 \notin dom(P_j)$ iff: $\{j\}$ is a full extension of σ.* *(rule 1)*
- *$1 \notin dom(P_j)$ iff:* $\begin{cases} |\mathcal{V}_{\mathcal{D}}^{\sigma^+}(j)| < \theta \quad \vee & \text{(rule 2)} \\ \exists k \in \sigma^-, \mathcal{V}_{\mathcal{D}}^{\sigma^+}(j) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(k). & \text{(rule 3)} \end{cases}$

Proof. Let σ be an extensible partial assignment and P_j be a free variable.

$0 \notin dom(P_j)$: ($\Rightarrow$) Let 0 be an inconsistent value. In this case, P_j can only take value 1. It means that $Clos(\sigma^+) = Clos(\sigma^+ \cup \{j\})$. Thus, $\mathcal{T}_{\mathcal{D}}(\sigma^+) = \mathcal{T}_{\mathcal{D}}(\sigma^+ \cup \{j\})$. By Definition 2, $\{j\}$ is a full extension of σ.
($\Leftarrow$) Let $\{j\}$ be a full extension of σ, which means that $Clos(\sigma^+) = Clos(\sigma^+ \cup \{j\})$ (Definition 2). The value 0 is inconsistent where j cannot be in σ^- (property 1).

$1 \notin dom(P_j)$: ($\Rightarrow$) Let 1 be an inconsistent value. This can be the case if the frequency of the current pattern σ^+ is set up below the threshold θ by adding the item j (i.e., $|\mathcal{V}_{\mathcal{D}}^{\sigma^+}(j)| < \theta$). Or, $\sigma^+ \cup \{j\}$ cannot be extended to a closed itemset: this is the case when there exists an item $k \in \sigma^-$ such that at each time the item j belongs to a transaction in the database, k belongs as well ($\mathcal{V}_{\mathcal{D}}^{\sigma^+}(j) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(k)$). Conversely, the lack of k (i.e., $k \in \sigma^-$) implies the lack of j as well. This means that: ($P_k = 0 \Rightarrow P_j = 0$).
($\Leftarrow$) This is a direct consequence of Proposition 1. □

The first rule takes its origin from item merging [15]. The second rule is a basic rule derived from the property of anti-monotonicity of the frequency (Property 1). To the best of our knowledge, the third rule is a new rule taking its originality from the reasoning made on absent items.

Example 4. Following Example 3, consider a partial assignment σ such that the variable P_1 is set to 0 (item A). That is, $\sigma^- = \{A\}$ and $\sigma^+ = \emptyset$. Value 1 from $dom(P_4)$ (item D) is inconsistent because the lack of A implies the lack of D in $\mathcal{D}$ (i.e., $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(D) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(A)$). Let now $P_1 = 0, P_4 = 0$, that is, $\sigma^- = \{A, D\}$ and $\sigma^+ = \emptyset$. If the variable P_3 is set to 1 (item C), value 1 from $P_i, i = \{2, 5, 6, 7\}$ (items B, E, F, G) is inconsistent because $|\mathcal{V}_{\mathcal{D}}^{\sigma^+}(P_i)| < 2$, and value 0 from P_8 (item H) is also inconsistent because $\{H\}$ is a full extension of σ.

4.2   CLOSEDPATTERN Filtering Algorithm

In this section, we present the algorithm FILTER-CLOSEDPATTERN (Algorithm 1) for enforcing domain consistency on the CLOSEDPATTERN constraint. FILTER-CLOSEDPATTERN incrementally maintains the internal data structures $\sigma = <\sigma^+, \sigma^-, \sigma^*>$ and the corresponding cover $\mathcal{T}_{\mathcal{D}}(\sigma^+)$. Using these two structures, one can check if an item is present or not in the vertical dataset $\mathcal{V}_{\mathcal{D}}$.

Algorithm 1 takes as input the vertical dataset $\mathcal{V}_{\mathcal{D}}$, a minimum support threshold θ, the current partial assignment σ on P where $\sigma^* \neq \emptyset$, and the variables P. As output, Algorithm 1 reduces the domains of P_i's and therefore, increases σ^+ and/or σ^-, and decreases σ^*.

The algorithm starts by checking if the current partial assignment is extensible or not (Proposition 1). This is performed by checking (1) if the size of the current cover is greater than the minimum support (line 4) and (2) if no item of a variable already instantiated to zero is a full extension of σ^+ (line 5).

Algorithm 1. FILTER-CLOSEDPATTERN($\mathcal{V}_{\mathcal{D}}, \theta, \sigma, P$)

---

1 Input: $\mathcal{V}_{\mathcal{D}}$: vertical database; θ : minimum support

2 InOut: $P = \{P_1 \dots P_n\}$: Boolean item variables; σ : current assignment.

3 begin

4 **if** $(|\mathcal{T}_{\mathcal{D}}(\sigma^+)| < \theta)$ **then return** *false*;

5 **if** $\exists i \in \sigma^- : |\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i)| = |\mathcal{T}_{\mathcal{D}}(\sigma^+)|$ **then return** *false*;

6 **foreach** $i \in \sigma^*$ **do**

7 | **if** $(|\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i)| = |\mathcal{T}_{\mathcal{D}}(\sigma^+)|)$ **then**

8 | | $dom(P_i) \leftarrow dom(P_i) - \{0\}$;

9 | | $\sigma^+ \leftarrow \sigma^+ \cup \{i\}$; $\sigma^* \leftarrow \sigma^* \setminus \{i\}$;

10 | **else if** $(|\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i)| < \theta)$ **then**

11 | | $dom(P_i) \leftarrow dom(P_i) - \{1\}$;

12 | | $\sigma^- \leftarrow \sigma^- \cup \{i\}$; $\sigma^* \leftarrow \sigma^* \setminus \{i\}$;

13 **foreach** $i \in \sigma^-$ **do**

14 | **foreach** $j \in \sigma^* : \mathcal{V}_{\mathcal{D}}^{\sigma^+}(j) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(i)$ **do**

15 | | $dom(P_j) \leftarrow dom(P_j) - \{1\}$;

16 | | $\sigma^- \leftarrow \sigma^- \cup \{j\}$; $\sigma^* \leftarrow \sigma^* \setminus \{j\}$;

17 **return** *true*;

   Lines 6–12 are a straightforward application of *rules 1* and *2* of Proposition 2. For each non-instantiated variable, (1) we check if value 0 is consistent: that item is not a full extension (lines 6–9), and (2) we check if value 1 is consistent: the new cover size by adding that item remains greater than θ (lines 10–12).

   Finally, lines 13–16 implement *rule 3* of Proposition 2. We prune value 1 from each free item variable $i \in \sigma^*$ such that its cover is a superset of the cover of an absent item $j \in \sigma^-$ $(\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(j))$.

Theorem 1. *Given a transaction database $\mathcal{D}$ of n items and m transactions, and a threshold minsup θ. Algorithm* FILTER-CLOSEDPATTERN *enforces domain consistency on the* CLOSEDPATTERN *constraint, or proves that it is inconsistent in time $O(n^2 \times m)$ with a space complexity of $O(n \times m)$.*

Proof. **DC:** FILTER-CLOSEDPATTERN implements exactly Proposition 1 and the three rules given in Proposition 2. Thus FILTER-CLOSEDPATTERN ensures domain consistency (see the description of Algorithm 1).

Time: Let $n = |\mathcal{I}|$ and $m = |\mathcal{T}|$. First, we need to compute $\mathcal{T}_{\mathcal{D}}(\sigma^+)$ which requires at most $O(n \times m)$. This is done only once. The cover $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i)$ can be computed by intersecting $\mathcal{T}_{\mathcal{D}}(\sigma^+)$ (already computed) and $\mathcal{T}_{\mathcal{D}}(\{i\})$ (given by the vertical representation) within at most $O(m)$. Checking *rules 1* and *2* on all free variables can be done in $O(n \times m)$ (lines 6–12). However, checking *rule 3* is cubic at lines 13–16 (i.e., $O(n \times (n \times m))$), where checking if a cover $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i)$ is a subset of another cover can be done in $O(m)$. Finally, the worst case complexity is $O(n \times (n \times m))$.

Space: The space complexity of FILTER-CLOSEDPATTERN lies in the storage of $\mathcal{V}_{\mathcal{D}}$, σ and the cover $\mathcal{T}$ data structures. The vertical representation $\mathcal{V}_{\mathcal{D}}$ requires at most $n \times m$ space. In the worst case, we have to store n items within σ and m transactions within $\mathcal{T}$. That is, the worst case space complexity is $O(n \times m + n + m) = O(n \times m)$. □

   During the solving process in depth first search, the whole space complexity is $O(n \times (m + n))$ because (1) the depth is at most n; (2) σ and $\mathcal{T}$ require $O(n \times (m + n))$; (3) the vertical representation is the same data used all along the solving process $O(n \times m)$; (4) $O(n \times (m + n)) + O(n \times m) = O(n \times (m + n))$.

Proposition 3 (Backtrack-free). *Extracting the total number of closed frequent patterns, noted C, is backtrack-free with a complexity in $O(C \times n^2 \times m)$ using* FILTER-CLOSEDPATTERN *to propagate the* CLOSEDPATTERN *constraint.*

Proof. FILTER-CLOSEDPATTERN ensures DC at each node of the search tree. Hence, the closed frequent patterns are guaranteed to be produced in a backtrack-free manner. The explored search tree is a binary full tree where each node is either a leaf (a solution) or possesses exactly two child nodes. The number of nodes is thus in $O(2 \times C)$. Knowing that ensuring DC is in $O(n^2 \times m)$, extracting the total number of closed frequent patterns is in $O(C \times n^2 \times m)$. □

4.3 Data Structures

To represent the transactional dataset and the cover of items, we adopted the vertical representation format [16]. Our implementation is in `or-tools` solver[3].

Static Structures: for each item $i \in \mathcal{I}$, $\mathcal{T}_\mathcal{D}(i) = \{t \in \mathcal{T} \mid i \in t\}$ is stored as a bitset of size m. If $t \in \mathcal{T}_\mathcal{D}(i)$, then the associated bit is set to 1 (0 otherwise).

Dynamic Structure: a vector $Memo$ of bitsets is used to store the cover of each partial solution. $Memo$ is a vector of size $n + 1$, since at the beginning of the search, the partial assignment is empty. Let σ be a partial assignment. Each time a (new) variable P_i is instantiated, the cover of the new partial assignment $\sigma \cup \{i\}$ is stored in $Memo$. If $P_i = 0$, the cover remains the same: $\mathcal{T}_\mathcal{D}(\sigma^+ \cup \{i\}) = \mathcal{T}_\mathcal{D}(\sigma^+)$. If $P_i = 1$, the cover of the new partial solution $\sigma \cup \{i\}$ is computed by a bitwise-AND between $\mathcal{T}_\mathcal{D}(\sigma^+)$ and $\mathcal{T}_\mathcal{D}(\{i\})$, and stored in $Memo$.

Backtracking. First, all P_i, as well as their domains $dom(P_i)$, are fully maintained by the `or-tools` backtracking. Then, a single value (the current index of the vector $Memo$) is asked to be managed by the `or-tools` backtracking. Each time a partial solution is extended (from σ to $\sigma \cup \{i\}$), the current index of $Memo$ is memorised. When a backtrack occurs (from $\sigma \cup \{i\}$ to σ), this value is restored by `or-tools` giving access to the cover of the (restored) partial solution.

Rule 3. The inclusion between two covers $\mathcal{V}_\mathcal{D}^{\sigma^+}(i) \subseteq \mathcal{V}_\mathcal{D}^{\sigma^+}(j)$ is rewritten as $\mathcal{V}_\mathcal{D}^{\sigma^+}(i) \cap \mathcal{V}_\mathcal{D}^{\sigma^+}(j) = \mathcal{V}_\mathcal{D}^{\sigma^+}(i)$, and the intersection is performed by a bitwise-AND.

5 Running Example

In this section, we illustrate the propagation of our CLOSEDPATTERN constraint and the difference that exists comparing to the use of a simple Reified Constraint Model (denoted by RCM, and detailed in Sect. 3). For that, let us take the transactional dataset given in Table 1: Five transactions t_1 to t_5 and eight items from A to H.

Figure 1 shows the tree search explored using CLOSEDPATTERN (**part(a)**) and the tree search explored using a reified model (**part(b)**) to extract closed frequent patterns at minimum support $\theta = 2$. Here, both approaches use the same branching heuristics, namely `Lex` on variables and `Max_val` on values. First of all, it is worth noticing that the search space that can be explored using CLOSEDPATTERN is defined only on decision variables (item variables), whereas the reified model adds a further dimension with auxiliary variables (transaction variables).

At the root node (node 1), no pruning is done since all items are frequent and no item is a full extension of the empty pattern (see Table 1). Thereafter, CLOSEDPATTERN and RCM are acting in the same manner on the branch $A = 1$. With $A = 1$, B, C, E, F, G, H become infrequent. That is, the 1 values are pruned (*rule 2*). With RCM on node 2, the pruning on the five decision variables (the
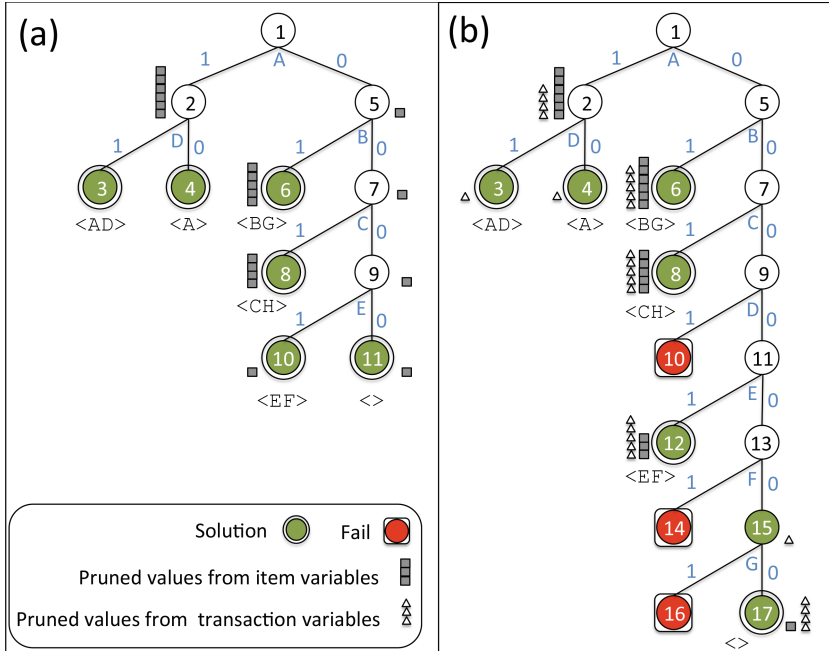

Fig. 1. (a) CLOSEDPATTERN and (b) Reified Constraint Model (RCM)

item variables) induce a pruning on four auxiliary variables (transaction variables). On the branch $A = 1$, two solutions are found: $\langle AD \rangle$ and $\langle A \rangle$.

Branching on $A = 0$ (node 5), the value 1 of D is pruned with *rule 3* of CLOSEDPATTERN. From Table 1, we have $D \Rightarrow A$, and a branching on $A = 0$ reduces D to 0. Here, we can say that the DC is maintained on node 5 using CLOSEDPATTERN, which is not the case using RCM. The same observation can be made on nodes 7, 9 and 11.

Let us take the node 6, here the branching on $A = 0$ and thereafter on $B = 1$ will make C, D, E, F, H infrequent (*rule 2*). Moreover, (*rule 1*) can be applied since G is a full extension of B (i.e., we cannot have a frequent closed pattern including B without G). That is, the value 0 is pruned from the domain of G, which allows us to reach the solution $\langle BG \rangle$. The same observation can be made on node 8.

To sum up, FILTER-CLOSEDPATTERN maintains DC at each node and thus, enumerates the solutions backtrack-free (no fails). The same cannot be said with RCM because the *rule 3* is never covered in this example and there are 3 fails.

6   Experiments

We made several experiments to compare and evaluate our global constraint with the state of the art methods (CP and specialized methods).

Benchmark Datasets. We selected several real and synthetic datasets [4,17] from FIMI repository[4] with large size. These datasets have varied characteristics representing different application domains. Table 2 reports for each dataset, the number of transactions $|\mathcal{T}|$, the number of items $|\mathcal{I}|$, the average size of transactions $|\widehat{\mathcal{T}}|$, its density ρ (i.e., $|\widehat{\mathcal{T}}|/|\mathcal{I}|$) and its *size* (i.e., $|\mathcal{T}| \times |\mathcal{I}|$). We note that the datasets are presented according to their size. They represent various numbers of transactions, numbers of items, densities. We have datasets that are very dense like Chess and Connect (resp. 49 % and 33 %), and others that are very sparse like Retail and BMS-Web-View1 (resp. 0.06 % and 0.5 %). Note that we have datasets of sizes going from $\approx 10^5$ to more than 10^9.

Table 2. Dataset Characteristics.

Dataset	$|\mathcal{T}|$	$|\mathcal{I}|$	$|\widehat{\mathcal{T}}|$	ρ	type of data	size
Chess	3 196	75	37	49%	game steps	239 700
Splice1	3 190	287	60	21%	genetic sequences	915 530
Mushroom	8 124	119	23	19%	species of mushrooms	966 756
Connect	67 557	129	43	33%	game steps	8 714 853
BMS-Web-View1	59 602	497	2.5	0.5%	web click stream	29 622 194
T10I4D100K	100 000	1 000	10	1%	synthetic dataset	100 000 000
T40I10D100K	100 000	1 000	40	4%	synthetic dataset	100 000 000
Pumsb	49 046	7 117	74	1%	census data	349 060 382
Retail	88 162	16 470	10	0.06%	retail market basket data	1 452 028 140

Experimental Protocol. The implementation of our approach was carried out in the `or-tools` solver. All experiments were conducted on an Intel Xeon E5-2680 @ 2.5 GHz with 128 Gb of RAM with a timeout of 3600*s*. For each dataset, we decreased the (relative) θ threshold until it is impossible to extract all closed patterns within the allocated time/memory. We have implemented two variants of CLOSEDPATTERN constraint: **(i)** CLOSEDPATTERN-DC ensuring DC with *rules 1,2 and 3* (cubic pruning). **(ii)** CLOSEDPATTERN-WC ensuring a weaker consistency with only *rules 1 and 2* (quadratic pruning). Comparisons are made with: **(i)** CP4IM, the state-of-the-art on CP approaches, that uses an RCM model. **(ii)** LCM, the state-of-the-art on specialized methods.

For CLOSEDPATTERN and CP4IM, we use the same branching heuristics, namely `Lex` on variables and `Max_val` on values. We experimented using the available distributions of `LCM-v5.3`,[5] and `CP4IM`,[6] with `Gecode` as the underlying solver of CP4IM. Table 3 gives a comparison between CLOSEDPATTERN (WC and DC versions), CP4IM and LCM. We report the number of closed patterns $\#C$ of each instance, the number of propagations, the number of nodes, the CPU times in seconds and the number of failures.

CLOSEDPATTERN (DC vs WC). Despite the pruning complexity, DC clearly dominates WC in terms of CPU times (except for *BMS1* dataset where both

---

[4] http://fimi.ua.ac.be/data/.

[5] http://research.nii.ac.jp/~uno/codes.htm.

[6] https://dtai.cs.kuleuven.be/CP4IM/.

are more or less equivalent). For instance, on the *pumsb* dataset with $\theta = 70\%$, DC is about 4 times faster than WC. As a second observation, the use of *rule 3* can reduce drastically the number of explored nodes and thus, number of propagations. For instance, we note a reduction of 38 % on explored nodes on *connect* and 98 % on *splice1* compared to WC.

CLOSEDPATTERN vs CP4IM. If we compare CLOSEDPATTERN with CP4IM, the main observation is that DC outperforms significantly CP4IM at all levels. In terms of CPU times and without counting the Out-of-memory instances, we can observe 23 instances (out of 30) with a speed-up factor between 2 and 15. Factors of 27 to 45 are noted for six instances and for one instance, we have a factor of 182. The weaker version (WC) is also better than CP4IM except two instances in *connect* and two instances in *chess*. In terms of number of propagations,

Table 3. CLOSEDPATTERN vs CP4IM vs LCM. (OOM: Out Of Memory; TO: TimeOut; (1): CLOSEDPATTERN-WC; (2): CLOSEDPATTERN-DC; (3): CP4IM)

D	θ	#C	#Propagations			#Nodes			Time (s)				Failures		
	(%)	(≈)	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	LCM	(1)	(2)	(3)
chess	50	10^5	4 037 618	920 377	9 201 740	4 037 211	738 901	738 907	7.26	3.21	10.97	0.32	1 649 155	0	3
	40	10^6	15 931 714	3 411 690	30 541 475	15 929 751	2 733 667	2 733 735	26.37	12.27	40.85	1.44	6 598 042	0	34
	30	10^6	71 744 915	13 298 679	105 040 377	71 734 057	10 679 631	10 681 739	109.36	45.92	136.31	6.07	30 527 213	0	1 054
	20	10^7	305 446 222	56 550 872	385 399 747	305 374 413	45 837 171	45 901 933	480.52	187.89	467.52	27.55	1.29×10^8	0	32 381
	10	10^8	1.51×10^9	304 335 522	1.66×10^9	1.51×10^9	247 960 091	249 411 325	2 287.98	969.40	1 950.51	141.55	6.34×10^8	0	725 671
splice1	20	10^2	54 055	25 078	913 779	53 991	487	487	0.10	0.59	22.59	0.04	26 752	0	0
	10	10^3	303 405	57 623	1 810 466	303 359	3 211	3 211	0.18	0.14	25.54	0.07	150 074	0	0
	5	10^4	5 013 077	1 706 269	36 211 067	5 013 031	63 935	63 935	3.39	3.23	138.54	1.46	2 474 548	0	0
	1	10^7	754 074 683	193 156 761	2.43×10^9	754 067 617	13 454 755	13 467 247	502.77	400.10	1 652.41	53.59	3.70×10^8	0	6 246
mushroom	30	10^3	6 323	1 530	430 016	6 179	853	1 039	0.46	0.64	0.77	0.00	2 663	0	93
	20	10^3	34 375	4 414	1 031 778	34 113	2 405	3 083	0.92	0.56	2.36	0.01	15 854	0	339
	10	10^4	148 429	17 456	2 771 719	147 623	9 793	13 281	0.38	0.15	5.32	0.04	68 915	0	1 744
	5	10^4	496 643	46 165	5 574 143	494 559	25 707	36 495	0.84	0.41	10.98	0.09	234 426	0	5 394
	1	10^5	2 347 734	178 834	13 813 312	2 339 109	103 343	168 999	3.42	1.74	24.06	0.22	1 117 883	0	32 828
	0.5	10^5	3 896 158	264 660	18 230 838	3 883 697	156 613	265 781	5.00	3.62	29.84	0.34	1 863 542	0	54 584
	0.1	10^5	7 148 533	516 041	31 222 435	7 130 827	328 233	529 289	9.96	5.40	41.38	0.47	3 401 297	0	100 528
	0.05	10^5	8 001 925	613 712	36 520 438	7 983 119	407 763	622 145	9.94	6.37	43.69	0.51	3 787 678	0	107 191
connect	90	10^3	12 377	10 272	1 865 236	9 725	6 973	6 973	2.45	0.92	7.10	0.22	1 376	0	0
	80	10^4	60 312	48 156	9 455 886	44 771	30 223	30 223	9.35	1.65	16.57	0.31	7 274	0	0
	70	10^4	158 817	120 184	24 969 205	115 335	71 761	71 761	24.26	4.09	33.72	0.40	21 787	0	0
	60	10^5	297 175	240 634	51 648 114	202 637	136 699	136 699	45.23	7.30	45.73	0.39	32 969	0	0
	50	10^5	570 870	469 442	98 602 318	378 603	260 223	260 223	85.99	14.53	110.19	0.52	59 190	0	0
	40	10^5	1 139 466	886 722	177 653 293	762 579	478 781	478 781	159.04	27.32	153.39	0.83	141 899	0	0
	30	10^5	2 173 593	1 752 199	313 288 691	1 400 609	920 823	920 823	263.64	49.97	304.52	1.37	239 893	0	0
	20	10^5	8 440 998	5 637 016	633 429 969	6 092 697	2 966 399	2 966 399	678.95	157.40	712.68	4.37	1 563 149	0	0
	10	10^7	55 152 559	31 370 763	1.76×10^9	41 641 659	16 075 555	16 075 555	3 110.90	760.71	2 597.89	17.70	12 783 052	0	0
BMS1	0.20	10^3	311 638	105 403	OOM	250 959	1 677	OOM	111.04	145.72	OOM	0.04	124 671	0	OOM
	0.16	10^3	493 069	116 902	OOM	432 419	2 617	OOM	161.60	218.88	OOM	0.06	214 939	0	OOM
	0.12	10^3	921 446	134 436	OOM	860 811	4 971	OOM	239.08	202.46	OOM	0.05	427 940	0	OOM
	0.08	10^3	3 638 431	200 049	OOM	3 577 735	20 787	OOM	2 441.84	1 899.70	OOM	0.11	1 778 490	0	OOM
T10*	10	1	1 000	1 000	OOM	1	1	OOM	0.46	0.24	OOM	0.03	0	0	OOM
	1	10^2	155 166	76 955	OOM	154 559	805	OOM	3.75	1.14	OOM	0.14	76 894	0	OOM
	0.5	10^3	509 852	174 015	OOM	509 441	2 185	OOM	10.29	3.11	OOM	0.42	253 647	0	OOM
	0.1	10^3	11 228 933	520 218	OOM	11 228 667	53 625	OOM	258.44	11.30	OOM	0.94	5 587 527	0	OOM
	0.05	10^4	21 004 412	1 257 876	OOM	21 003 915	93 987	OOM	477.22	27.96	OOM	1.30	10 454 964	0	OOM
	0.01	10^5	130 686 806	17 313 642	OOM	130 673 093	566 795	OOM	1 471.32	905.17	OOM	3.49	65 053 149	0	OOM
T40*	10	10^3	8 712	4 900	OOM	7 801	177	OOM	0.89	1.13	OOM	0.43	3 818	0	OOM
	5	10^2	97 982	47 983	OOM	97 289	643	OOM	2.34	1.78	OOM	1.31	48 328	0	OOM
	1	10^5	19 932 041	937 766	OOM	19 931 799	130 477	OOM	599.49	25.78	OOM	21.32	9 900 663	0	OOM
	0.5	10^6	TO	6 394 931	OOM	TO	2 551 883	OOM	TO	953.58	OOM	13.31	TO	0	OOM
pumsb	95	10^2	7 981	7 438	OOM	877	223	OOM	2.85	1.45	OOM	0.20	328	0	OOM
	90	10^3	17 429	11 074	OOM	10 323	2 933	OOM	44.92	13.37	OOM	0.29	3 695	0	OOM
	85	10^4	61 459	29 149	OOM	54 331	17 027	OOM	155.79	58.84	OOM	0.27	18 652	0	OOM
	80	10^4	176 955	89 805	OOM	169 743	66 615	OOM	640.59	133.97	OOM	0.33	51 564	0	OOM
	75	10^5	472 624	249 382	OOM	465 267	202 165	OOM	1 010.43	271	OOM	0.48	131 551	0	OOM
	70	10^5	969 374	567 929	OOM	961 763	482 517	OOM	2 150.80	509.79	OOM	0.69	239 623	0	OOM
retail	10	10^2	16 501	16 490	OOM	37	19	OOM	1	2.55	OOM	0.06	9	0	OOM
	1	10^2	31 089	19 852	OOM	14 695	329	OOM	11.03	4.02	OOM	0.10	7 188	0	OOM
	0.5	10^3	205 688	56 868	OOM	189 475	1 233	OOM	61.41	12.73	OOM	0.32	94 156	0	OOM
	0.1	10^4	23 520 823	4 506 219	OOM	23 506 749	15 901	OOM	495.52	796.82	OOM	0.80	11 745 679	0	OOM
	0.05	10^4	114 357 067	18 342 468	OOM	114 345 005	40 229	OOM	2 352.14	2 645.06	OOM	1.07	57 152 804	0	OOM

we observe a gain factor within a range from 13 to 300. This is because of
the huge number of propagator calls for reified constraints comparing to one
propagator call using CLOSEDPATTERN. The number of explored nodes is also
reduced, sometimes by half (e.g., *mushroom*), using CLOSEDPATTERN-DC. This
is not a surprise as its number of explored nodes is optimal. Another observation
is the experimental validation of Proposition 3: CLOSEDPATTERN-DC extracts
the closed patterns in a *backtrack-free* manner. All solutions are enumerated
without any fail (see failures column in Table 3). CP4IM requires an important
number of backtracks on most datasets. On *connect* and three instances of *splice1*
we can observe that CLOSEDPATTERN-DC and CP4IM explore the same number
of nodes. The reified model on such dense datasets and using (Lex\Max_val)
heuristics is able to prune all inconsistent values. This result is confirmed by the
number of failures always equal to zero (backtrack-free). Even if the WC version
is faster, CP4IM remains better in terms of pruning (#nodes). Finally, the major
drawback using the reified model for frequent pattern extraction is the memory
consumption. We denote 25 Out-of-memory (out of 51 instances). The Out-of-
memory state is due to the huge number of reified constraints. For instance, if
we take the *T40I10D100K* dataset, the CP model produced by CP4IM contains
$|\mathcal{T}| + |\mathcal{I}| = 101\,000$ variables, $|\mathcal{T}| = 100\,000$ reified constraints to express the
channeling constraints, $2 \times |\mathcal{I}| = 2 \times 1\,000$ reified constraints to express the
closure and frequency constraints. This means that the CP solver has to load
in memory a CP model of 102 000 reified constraints expressed on 101 000
variables, which represents the *size* given in Table 2. From Table 2, we observe
that Gecode is not able to handle CP4IM models on datasets of size greater than
$\approx 10^7$ (greater than *connect* dataset).

CLOSEDPATTERN vs LCM. In terms of CPU times, LCM remains the leader on
basic queries. However, CLOSEDPATTERN is quite competitive as a declarative
approach. For instance, if we take *chess* dataset, LCM is 15 times faster than
CLOSEDPATTERN-DC on average, where it is more than 120 times faster on
average comparing to CP4IM. CLOSEDPATTERN pruning acts only within the
current node of the search tree, without imposing any condition on the main
search algorithm such as variable or value orderings. This allows to consider
new constraints and let the main search algorithm adopt the best heuristics
favouring the whole solving. To illustrate our point, we propose to model a
particular problem (*k*-pattern sets) in a declarative manner where LCM could not
meet this need.

k-**Patterns Instance.** A promising road to discover useful patterns is to impose
constraints on a set of *k* related patterns (*k*-pattern sets) [5,9]. In this setting,
the interest of a pattern is evaluated w.r.t. a set of patterns. We propose to
model and solve a particular instance, coined `dist_kpatterns_lb_ub`. Here, we
aim at finding *k* closed patterns $\{P^1, \ldots, P^k\}$ s.t:

(i) $\forall i \in [1, k] :$ CLOSEDPATTERN(P^i) (Closed Frequent Patterns),
(ii) $\forall i, j \in [1, k] : P^i \cap P^j = \emptyset$ (all distinct patterns constraints),
(iii) $\forall i \in [1, k] :$ `lb` $< |P^i| <$ `ub` (min and max size constraints).

Figure 2 shows a comparison between two models, M1 using CLOSEDPATTERN for the CFPM part of the problem, M2 using a CP4IM implementation. We selected two dataset instances where CP4IM does not reach the Out-of-memory state and where we have a reasonable number of closed patterns, *chess* with $\theta = 80\,\%$ (5084 closed patterns) and *connect* with $\theta = 90\,\%$ (3487 closed patterns), and we have varied k with a timeout of $3600\,s$. After a few preliminary tests, the bounds lb and ub on the size of the patterns were set to 2 and 10 respectively. On *chess*, model M1 is robust and scales well: it is linear on k and never exceeds 6 min even with $k = 12$ ($323.53\,s$). M2 follows an exponential scale and goes beyond the timeout with only $k = 8$ ($7222.41\,s$). The same observation can be made on the **connect** instance, but in a more pronounced way on the exponential scale followed by M2. With $k = 4$, M2 goes beyond the timeout with $5428.05\,s$ whereas M1 confirms its linear behavior when varying k from 2 to 12.

For such problems, a baseline can be the use of specialized methods with post-processing. One can imagine **(i)** the use of LCM to extract the total number of closed patterns and **(ii)** a generate-and-test search trying to find distinct patterns of a given size. Such approach can be very expensive. Here, the postprocessing will generate all the possible k combinations of closed patterns. For instance, we recall that for *chess* with $\theta = 80\,\%$ we have 5084 closed patterns. With $k = 12$ and using M1, we need less than 6 min, where using the baseline we have to cope with a massive number of combinations. Thus, this last experiment confirms that if LCM is faster on basic queries (e.g., asking for closed frequent with given size), it cannot cope with complex queries. It would need to think and to propose an adhoc solution whereas CP enables a novice DM-user to express his query as constraints.

7   Conclusion

In this paper we have introduced a new global constraint for Closed Frequent Pattern Mining. The CLOSEDPATTERN constraint captures the particular semantic of the CFPM problem, namely the minimum frequency and closedness of patterns. To propagate efficiently this global constraint, we have first defined three

chess($\theta = 80\%$, lb $= 2$, ub $= 10$)   connect ($\theta = 90\%$, lb $= 2$, ub $= 10$)
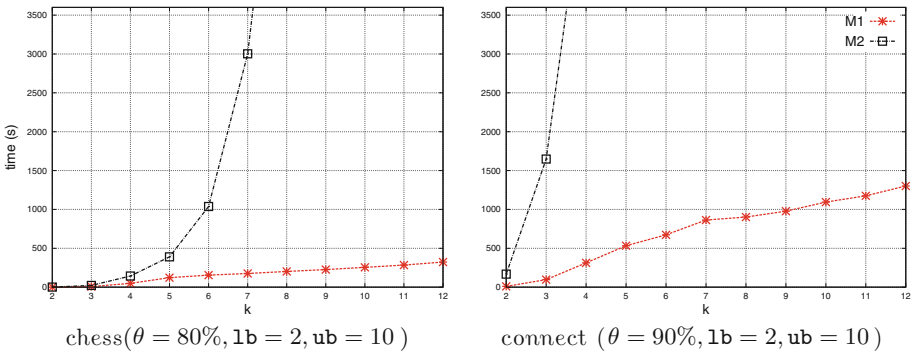
Fig. 2. dist_kpatterns_lb_ub instance using CLOSEDPATTERN and CP4IM.

filtering rules that ensure domain consistency. Second, we have defined a filtering algorithm that establishes domain consistency in a *cubic* time complexity and *quadratic* space complexity. We have implemented this filtering algorithm into the `or-tools` solver using a vertical representation of datasets and smart data structures. We have conducted an experimental study on several real and synthetic datasets, showing the efficiency and the scalability of the global constraint compared to a reified constraints approach such as `CP4IM`. Finally, to show the applicability and the flexibility of CLOSEDPATTERN compared to specialized methods we performed experiments on an instance of k-pattern set problem where CLOSEDPATTERN is integrated with a set of constraints.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., 26-28 May 1993, pp. 207–216. ACM Press (1993)
2. Brin, S., Motwani, R., Silverstein, C.: Beyond market baskets: generalizing association rules to correlations. In: SIGMOD, pp. 265–276 (1997)
3. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 204–212. ACM (2008)
4. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using FP-Trees. IEEE Trans. Knowl. Data Eng. **17**(10), 1347–1362 (2005)
5. Guns, T., Nijssen, S., De Raedt, L.: k-pattern set mining under constraints. IEEE Trans. Knowl. Data Eng. **25**(2), 402–418 (2013)
6. Guns, T., Nijssen, S., De Raedt, L.: Itemset mining: a constraint programming perspective. Artif. Intell. **175**(12), 1951–1983 (2011)
7. Hoeve, W., Katriel, I.: Global constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, pp. 169–208. Elsevier Science Inc., New York (2006)
8. Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., Charnois, T.: PREFIX-PROJECTION global constraint for sequential pattern mining. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 226–243. Springer, Heidelberg (2015)
9. Khiari, M., Boizumault, P., Crémilleux, B.: Constraint programming for mining n-ary patterns. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 552–567. Springer, Heidelberg (2010)
10. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Min. Knowl. Discov. **1**(3), 241–258 (1997)
11. Nijssen, S., Guns, T.: Integrating constraint programming and itemset mining. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010, Part II. LNCS, vol. 6322, pp. 467–482. Springer, Heidelberg (2010)
12. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Efficient mining of association rules using closed itemset lattices. Inf. Syst. **24**(1), 25–46 (1999)
13. Pei, J., Han, J., Mao, R.: CLOSET: an efficient algorithm for mining frequent closed itemsets. In: SIGMOD Workshop on Data Mining and Knowledge Discovery, pp. 21–30 (2000)
14. Uno, T., Asai, T., Uchida, Y., Arimura, H.: An efficient algorithm for enumerating closed patterns in transaction databases. In: DS 2004, pp. 16–31 (2004)

15. Wang, J., Han, J., Pei, J.: CLOSET+: searching for the best strategies for mining frequent closed itemsets. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 236–245 (2003)
16. Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. In: SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 326–335 (2003)
17. Zaki, M.J., Hsiao, C.: CHARM: an efficient algorithm for closed itemset mining. In: SIAM International Conference on Data Mining, pp. 457–473 (2002)

Clique and Constraint Models for Maximum Common (Connected) Subgraph Problems

Ciaran McCreesh[1(✉)], Samba Ndojh Ndiaye[2], Patrick Prosser[1], and Christine Solnon[3]

[1] University of Glasgow, Glasgow, Scotland
c.mccreesh.1@research.gla.ac.uk
[2] Université Lyon 1, LIRIS, UMR5205, 69621 Villeurbanne, France
[3] INSA-Lyon, LIRIS, UMR5205, 69621 Villeurbanne, France

Abstract. The maximum common subgraph problem is to find the largest subgraph common to two given graphs. This problem can be solved either by constraint-based search, or by reduction to the maximum clique problem. We evaluate these two models using modern algorithms, and see that the best choice depends mainly upon whether the graphs have labelled edges. We also study a variant of this problem where the subgraph is required to be connected. We introduce a filtering algorithm for this property and show that it may be combined with a restricted branching technique for the constraint-based approach. We show how to implement a similar branching technique in clique-inspired algorithms. Finally, we experimentally compare approaches for the connected version, and see again that the best choice depends on whether graphs have labels.

1 Introduction

Maximum common subgraph problems arise in biology and chemistry [16,20,40], in computer vision [7,9], in the analysis of source code [12], binary programs [19], and circuit designs [9], in character recognition problems [27], and in many other domains [49], both directly and as a way of measuring the similarity or difference between two graphs [5,18,23]. We illustrate two variants of this problem in Fig. 1—in both cases we are finding an induced subgraph and maximising the number of vertices selected, but in the second variant the common subgraph must be connected.

1.1 Definitions and Notation

We introduce definitions and algorithms on undirected and unlabelled graphs; the extension to general graphs is straightforward and is discussed in Sect. 2.3.

Fig. 1. A maximum common induced subgraph of the first two graphs has eight vertices, shaded. However, if we require the common subgraph to be connected, only seven vertices may be selected—one way to do this is shown in the third and fourth graphs.

An undirected graph G is defined by a finite set of vertices $\mathrm{V}(G)$ and a set of undirected edges $\mathrm{E}(G) \subseteq \mathrm{V}(G) \times \mathrm{V}(G)$, where $(u,v) \in \mathrm{E}(G) \Rightarrow (v,u) \in \mathrm{E}(G)$. The neighbourhood of a vertex v, written $\mathrm{N}(G,v)$, is the set of vertices to which it is adjacent, so $\mathrm{N}(G,v) = \{u \in \mathrm{V}(G) : (u,v) \in \mathrm{E}(G)\}$. Given a graph G, two vertices $v_s, v_e \in \mathrm{V}(G)$ are connected by a path in G if there exists a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $v_0 = v_s$, $v_k = v_e$, and each pair of successive vertices is connected by an edge, i.e. $\forall i \in \{1 \ldots k\}, (v_{i-1}, v_i) \in \mathrm{E}(G)$. A graph G is *connected* if every distinct pair of vertices is connected by a path.

The *subgraph* of a graph G *induced by* a set $H \subseteq \mathrm{V}(G)$, written $G[H]$, is the graph with vertex set H, and with every edge in G which has both endpoints in G, i.e. $\mathrm{E}(G[H]) = \mathrm{E}(G) \cap (\mathrm{V}(H) \times \mathrm{V}(H))$. We will consider only subgraphs which are induced by some set. It is also possible to permit removing edges which are not incident to removed vertices, thus leading to partial subgraphs—we do not consider this possibility in this paper, although everything we discuss may be extended to the partial case [34,54].

A graph G is *isomorphic* to another graph H if there exists a bijective function $f : \mathrm{V}(G) \rightarrow \mathrm{V}(H)$ which preserves edges and non-edges, i.e. $\forall (u,v) \in \mathrm{V}(G) \times \mathrm{V}(G), (u,v) \in \mathrm{E}(G) \Leftrightarrow (f(u), f(v)) \in \mathrm{E}(H)$. A *common subgraph* of two graphs G and H is a graph isomorphic to subgraphs of both G and H. A *common connected subgraph* is a common subgraph which is connected. A *Maximum Common Subgraph*, or MCS (resp. *Maximum Common Connected Subgraph*, or MCCS) is a common subgraph (resp. common connected subgraph) which has a maximum number of vertices.

1.2   Overview

In Sect. 2, we review existing approaches for solving the MCS problem, with a specific focus on Constraint Programming (CP)-based techniques, and on reductions of the problem to finding a maximum clique in an association graph.

Previous experimental evaluations have used simple maximum clique algorithms, or even enumeration algorithms—for example, Vismara and Valery [54] compare a modified form of the Bron-Kerbosch maximal clique enumeration algorithm [3] with a CP optimisation approach. Our experience suggests that a modern maximum clique algorithm could give many orders of magnitude improvement, due to a strong bound function which prunes the search space. Therefore, in Sect. 3, we re-evaluate the clique-based approach using a modern

algorithm, and we show that it outperforms CP on labelled graphs, and that it is competitive with CP on unlabelled graphs, contradicting earlier conclusions.

Then, in Sect. 4, we consider the MCCS problem. For the CP approach, we may add a global connectedness constraint to the model. Alternatively, we may use a special branching rule [54] to grow connected subgraphs only. These two techniques may be combined, and we experimentally show that the best results are in fact obtained when combining them. When solving the MCCS problem with a clique-based approach, neither technique seems directly viable with an association graph encoding. However, we show that it is possible to adapt the combined branching and bounding rule used by modern clique algorithms to maintain connectedness during search. We compare the clique-based approach with the best CP variant for MCCS, and we show that it outperforms CP on labelled graphs, whereas it is outperformed by CP on unlabelled graphs.

2   Existing Complete Approaches for MCS

There are two main approaches for solving MCS. The first approach (described in Sect. 2.1) is based on CP, whilst the second (described in Sect. 2.2) is based on a reformulation of MCS into a maximum clique problem. Both approaches are described for undirected, unlabelled graphs; their extension to richer graphs is discussed in Sect. 2.3. Other approaches have been tried, mixed integer programming [37] and heuristics [17]; SAT encodings seem to struggle even for subgraph isomorphism [31].

2.1   Constraint Programming Models for MCS

McGregor [32] proposed a branch and bound algorithm: each branch of the search tree corresponds to the matching of two vertices, and a bounding function evaluates the number of vertices that still may be matched so that the current branch is pruned as soon as this bound becomes lower than the size of the largest known common subgraph. CP approaches may be viewed as enhancements of this branch and bound algorithm.

Vismara and Valery [54] introduced the first explicit CP model. Given two graphs G and H, this model associates a variable x_v with every vertex v of G, and the domain of this variable contains all vertices of H, plus an additional value $\perp$: variable x_v is assigned to $\perp$ if vertex v is not matched to any vertex of H; otherwise x_v is assigned to the vertex of H to which it is matched. Edge constraints are introduced in order to ensure that variable assignments preserve edges and non-edges between matched vertices, i.e. $\forall u, v \in V(G), (x_u = \perp) \vee (x_v = \perp) \vee ((u, v) \in E(G) \Leftrightarrow (x_u, x_v) \in E(H))$. Difference constraints are introduced in order to ensure that each vertex of H is assigned to at most one variable, i.e. $\forall u, v \in V(G)$ distinct, $(x_u = \perp) \vee (x_v = \perp) \vee (x_u \neq x_v)$.

This CP model was improved by Ndiaye and Solnon [34] by replacing binary difference constraints with a soft global allDifferent constraint which maximizes the number of x_u variables that are assigned to values different from $\perp$, while
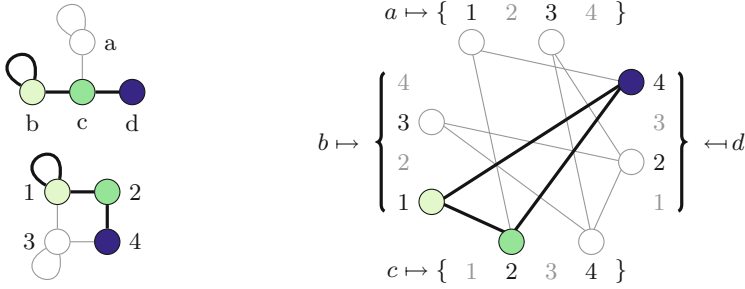
Fig. 2. A maximum common induced subgraph between the two graphs on the left has three vertices—one solution is highlighted. On the right, the association graph encoding: the highlighted clique of size three shows the same solution. The "missing" vertices correspond to assignments which are impossible due to the presence or absence of loops.

ensuring they are all different when they are not assigned to $\perp$. They find that the combination "MAC+Bound" (resp. "FC+Bound") obtains the best results on labelled (resp. unlabelled) graphs and outperforms the two previous approaches. The combination "MAC+Bound" maintains arc consistency [41] of edge constraints, whereas the combination "FC+Bound" simply performs forward-checking on these constraints. In both combinations, the "Bound" filtering checks whether it is possible to assign distinct values to enough x_u variables to surpass the best cost found so far—it is a weaker version of GAC($softAllDiff$) [36] which computes the maximum number of variables that can be assigned distinct values.

2.2   Reformulation of MCS to a Maximum Clique Problem

An alternative approach to MCS is to reduce the problem to finding a maximum clique in an association graph [2,15,24,40]. An association graph (or compatibility graph, or weak modular product) of two graphs G and H is an undirected graph $G \triangledown H$ with vertex set $V(G \triangledown H) = \{(v, v') \in V(G) \times V(H) : (v, v) \in E(G) \Leftrightarrow (v', v') \in E(H)\}$—to avoid confusing vertices of $G \triangledown H$ with vertices of the two original graphs, we call vertices of $G \triangledown H$ *matching nodes*, as each vertex (u, u') of $G \triangledown H$ denotes the matching of u with u'. The edges of $G \triangledown H$ connect matching nodes which denote compatible assignments, so two matching nodes (u, u') and (v, v') are adjacent if $u \neq v$ and $u' \neq v'$, and if they preserve both edges and non-edges, so $(u, v) \in E(G) \Leftrightarrow (u', v') \in E(H)$. We illustrate this in Fig. 2.

A *clique* is a subgraph whose vertices are all pairwise adjacent. A clique is *maximal* if it is not strictly included in any other clique, and it is *maximum* if it is a largest clique of a given graph, with respect to the number of vertices. A clique in an association graph corresponds to a set of compatible matchings. Therefore, such a clique corresponds to a common subgraph, and a maximum

clique of $G \triangledown H$ is an MCS of G and H. It follows that any method able to find a maximum clique in a graph can be used to solve the MCS problem.

Note that the association graph is a partial subgraph of the microstructure [21] associated with the CP model of Vismara and Valery [54]: the microstructure has more matching nodes than the association graph because it has a matching node $(u, \perp)$ for each vertex u of G. Each clique of size $|V(G)|$ in the microstructure corresponds to a common subgraph, the size of which is defined by the number of matching nodes that do not contain $\perp$.

2.3   Extension to Labelled or Directed Graphs

In some applications, labels may be associated with vertices or edges. We denote $\lambda(u)$ and $\lambda((u, v))$ the label of a vertex u and an edge (u, v), respectively. Where graphs are labelled, any isomorphism f must additionally preserve labels, so we require $\lambda(f(v)) = \lambda(v)$ for any vertex v, and $\lambda((f(u), f(v))) = \lambda((u, v))$ for any edge (u, v). This kind of label compatibility constraint is handled in a straightforward way in both CP and clique-based approaches to MCS. For CP, we restrict the domain of every variable x_u to vertices with compatible labels, and ensure that edge labels are preserved in edge constraints. For clique-based approaches, label compatibility is handled through the definition of the association graph, by restricting the set of matching nodes to pairs of vertices with compatible labels, and the set of matching edges to pairs of edges with compatible labels.

The extension of MCS algorithms to directed graphs, where isomorphisms must preserve directed edges, is similarly straightforward.

Labels and directed edges usually simplify the solution process, both for CP and clique-based approaches: vertex labels reduce domain sizes for CP, and the number of matching nodes in association graphs; edge labels tighten edge constraints for CP, and make the association graph sparser for clique-based approaches. It is worth noting that edge constraints do not help CP approaches to do more filtering so long as $\perp$ remains in variable domains: every pair of variables (x_i, x_j) having $\perp \in D(x_j)$ is arc consistent, since $\perp$ is a support for any value $u \in D(x_i)$. However, as soon as $\perp$ is removed from domains (i.e. when the number of variables assigned to $\perp$ has reached the best known bound on the size of the MCS), maintaining arc consistency may filter values, and then tighter constraints increase the opportunities for filtering.

3   Re-evaluating the Clique Model for MCS

Previous experimental evaluations of the association graph model have used either maximal clique enumeration algorithms [22,54] (even when the maximisation problem was being considered), or very simple maximum clique algorithms [6,8], and so their conclusions may now be overly pessimistic. Thus we re-evaluate the approach using a modern maximum clique algorithm. Association graphs are dense, even if the input is sparse, so we will using (the single-threaded,
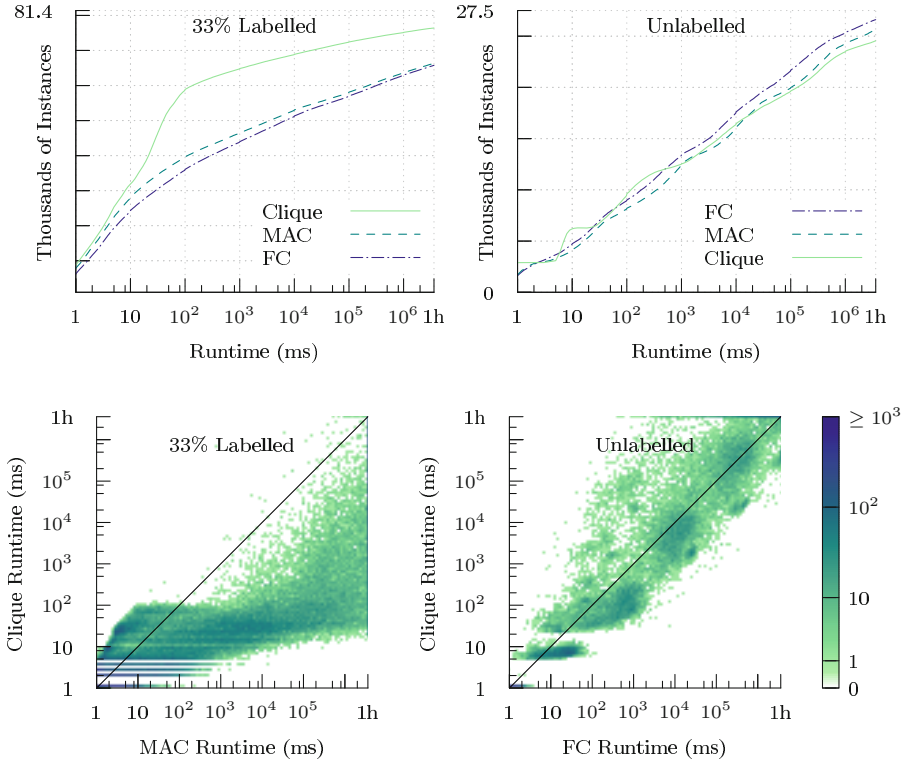
Fig. 3. The cumulative number of MCS instances solved in under a certain time: on the top, 33 % labelled graphs, and then unlabelled and undirected graphs. On the bottom, an instance-by-instance comparison of the clique model with the best CP model, with 33 % labelled graphs (with MAC) on the left, and unlabelled and undirected graphs (with FC) on the right.

bit-parallel version of) the maximum clique solver by McCreesh and Prosser [30], which implements Prosser's [38] "MCSa1" variant of a series of algorithms due to Tomita *et al.* [51–53], using a bitset encoding due to San Segundo *et al.* [45,47]. We compare this to the "FC+Bound" and "MAC+Bound" (simply referred to as FC and MAC) CP implementations of Ndiaye and Solnon [34], using smallest domain first for variable ordering, and a value ordering which prefers vertices of most similar degree. We perform our experiments on machines with Intel Xeon E5-2640 v2 CPUs and 64GBytes RAM; software was compiled using GCC 4.9, and a timeout of one hour was used.

We consider a randomly generated database [8,42] commonly used for benchmarking maximum common subgraph problems. The dataset contains different classes of graphs: randomly connected graphs with different densities; 2D, 3D, and 4D regular and irregular meshes; regular bounded valence graphs, and irregular bounded valence graphs. For each pair of graphs, there are 3 different

labellings such that the number of different labels is equal to 33 %, 50 % or 75 % of the number of vertices. In this paper, we report experiments with unlabelled graphs (labels are ignored), and with 33 % labellings (the problem becomes very easy with larger numbers of labels). For unlabelled graphs, we select the 27,500 graph pairs where the number of vertices in each graph is no more than 35; for labelled graphs, which we find less computationally challenging, we select all 81,400 graph pairs, to include graphs with up to 100 vertices.

The two plots on the top of Fig. 3 display the cumulative number of instances solved with respect to time. When graphs are labelled, the clique-based approach clearly outperforms either CP model, and MAC has a slight advantage over FC. (Recall that edge labels decrease the density of the association graph, which is typically very beneficial for clique algorithms, but do not help CP until $\perp$ is removed from domains.) For unlabelled graphs, the three approaches are broadly comparable, and ultimately FC beats MAC, which beats the clique approach. The bottom row gives a per-instance comparison of the best CP approach with the clique approach: the heatmaps are similar to scatter plots, but due to the large number of instances, we colour each point according to the density of solutions around that point. For labelled graphs, the clique approach comes close to dominating MAC on non-trivial instances (which suggests that there is unlikely to be scope for per-instance algorithm selection here). For unlabelled graphs, there is still a broad correlation between the runtimes; the clique approach rarely wins by more than one order of magnitude, but is sometimes much worse.

A closer inspection of the data suggests that the different randomness models used to generate instances have little effect on the runtimes for either approach. However, the relative size of the solution matters, particularly for the clique algorithm: if the solution is large (i.e. the two input graphs are very similar), the clique approach finds nearly every labelled instance trivial.

4   Finding Maximum Common Connected Subgraphs

In many applications, the common subgraph must satisfy some additional constraints. This is usually handled in a straightforward way in CP, by branching rules and/or constraint propagation. In clique-based approaches, some constraints may be handled by modifying the definition of the association graph—for example, constraints on pairs of vertices that may be matched are handled by removing inconsistent pairs from $\mathrm{V}(G \triangledown H)$. However, more global constraints cannot be handled by modifying the definition of the association graph.

In this paper, we focus on the connectedness constraint, which occurs in many applications [16, 22, 40, 54]. Adding the connectedness requirement makes certain special cases solvable in polynomial time, including outerplanar graphs of bounded degree [1] and trees [14], but the general case remains NP-hard. As illustrated in Fig. 1, the MCCS cannot be deduced from the MCS: we need to ensure connectedness during search. In Sect. 4.1, we show that in CP this may be done in two different ways that may be combined, and we show in Sect. 4.2 that the best results are obtained when combining them. In Sect. 4.3, we introduce
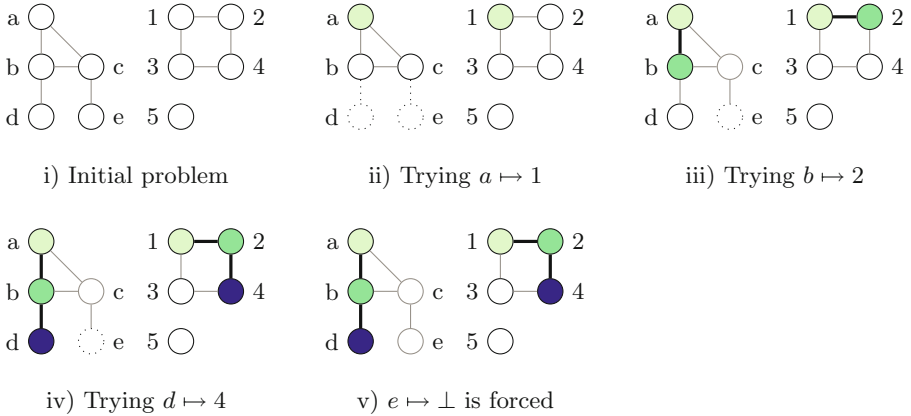
i) Initial problem

ii) Trying $a \mapsto 1$

iii) Trying $b \mapsto 2$

iv) Trying $d \mapsto 4$

v) $e \mapsto \bot$ is forced

Fig. 4. Suppose we are looking for a connected common subgraph, using the graph on the left for variables and the graph on the right (which has an isolated vertex) for values. We initially consider $a \mapsto 1$. Our restricted branching rule requires us to select either variable b or variable c subsequently, not d or e. We try $b \mapsto 2$, which adds d to the branchable variables, and forces $c \mapsto \bot$. We may now only branch on d, and we try $d \mapsto 4$. Now the only remaining variable is unbranchable, and so $e = \bot$ is forced, even though 5 remains in its domain and does not violate any constraints.

a new way for ensuring connectedness in a clique-based approach. Finally, we compare CP and our clique-based approach in Sect. 4.4.

For MCCS we consider only undirected graphs (and so we treat directed edges in the inputs as being undirected). For directed graphs, there is more than one notion of connectivity, and it is not clear which should be selected—the approaches we consider extend easily to weakly connected directed graphs, but not to the strongly connected case (for which we know of no applications).

4.1   Ensuring Connectedness in CP

Vismara and Valery [54] implement the connectedness constraint by using a branching rule which selects the next variable to be assigned. Let A be the set of variables already assigned to values different from $\bot$. The next variable to be assigned is chosen within the set of unassigned variables which are the neighbour of at least one vertex of A. When this set is empty, all remaining unassigned variables are assigned to $\bot$. We illustrate this in Fig. 4.

A more traditional CP approach would be to express connectedness as a conventional constraint. For example, CP(Graph) [13] introduces graph domain variables and enforces connectivity via the *reachable* constraint, ensuring that there is a path from a specified vertex to a specified set of vertices. One such constraint could be posted for each of the vertices in the graph, encoding the transitive closure of the graph. Brown et al. [4] explored the use of constraint programming in the generation of connected graphs with specified degree sequences.

Two constraints were combined: the graphical constraint (a backtrackable implementation of the Havel-Hakimi algorithm), and a connectivity constraint implemented using sets of vertices, where vertex sets A and B are combined when there exists a pair of vertices $v \in A$ and $w \in B$ and an edge $(v, w) \in E$. Residual degree counts are maintained on components and vertices to enforce graphicality and connectivity. Prosser and Unsworth [39] proposed a connectivity constraint for connected graph generation where decision variables are edges (the search process accepts and rejects edges). The constraint employed depth first search to maintain the set of tree edges and back edges, associating path counters on these edges. The counters were then used to detect the existence of cut-edges and protects these by forcing edges.

In all these previous works, the goal was to ensure that a given set of vertices is connected. For MCCS, the problem is slightly different: we have to ensure that the number of connected vertices that may be matched (in both graphs) is greater than the size of the largest common subgraph previously found. Therefore, we introduce a new filtering algorithm to ensure connectedness consistency for MCCS. Let us consider two graphs G and H, and let D be the current domains (we suppose that $D(x_u)$ is a singleton when x_u is assigned). Let S and T be the sets of vertices of G and H respectively which may belong to the common subgraph, i.e. $S = \{u \in V(G) : D(x_u) \neq \{\bot\}\}$, and $T = \cup_{u \in V(G)} D(x_u) \setminus \{\bot\}$. Connectedness consistency ensures that both $G[S]$ and $H[T]$ are connected graphs.

Connectedness consistency is ensured only once a first variable has been assigned, rather than at the root of search. Let x_u be the first assigned variable, and v the value assigned to x_u. To ensure connectedness consistency, we perform a traversal of G (resp. H), starting from u (resp. v), and we initialize S (resp. T) with all visited vertices. Then, for each vertex $v \in V(G) \setminus S$, we set x_v to $\bot$, and for each $w \in V(H) \setminus T$, we remove w from all domains to which it belongs.

During search, each time a variable is assigned to $\bot$, we remove the corresponding vertex from S and perform a new traversal of $G[S]$ starting from the initial vertex u. For each vertex $w \in S$ that is not visited by the traversal, we remove w from S and assign x_w to $\bot$. Also, each time a value is removed from a domain so that this value no longer belongs to any domain, we remove it from T, and perform a new traversal of $H[T]$ starting from the initial vertex v. For each vertex w that is not visited by the traversal, we remove w from T, and remove w from all domains to which it belongs.

Finally, the two approaches for ensuring connectedness (branching and filtering) are complementary and may be combined: at each step of the search, we select the next variable to be assigned within the neighbors of A, and each time a vertex of H is removed from a domain we filter domains to ensure connectedness consistency. In the example in Fig. 4, after the first assignment, filtering alone would remove 5 from every domain but would allow branching on any remaining variable, whilst branching alone would force the next variable to be either b or c but would not immediately eliminate 5 from the domains of d and e.
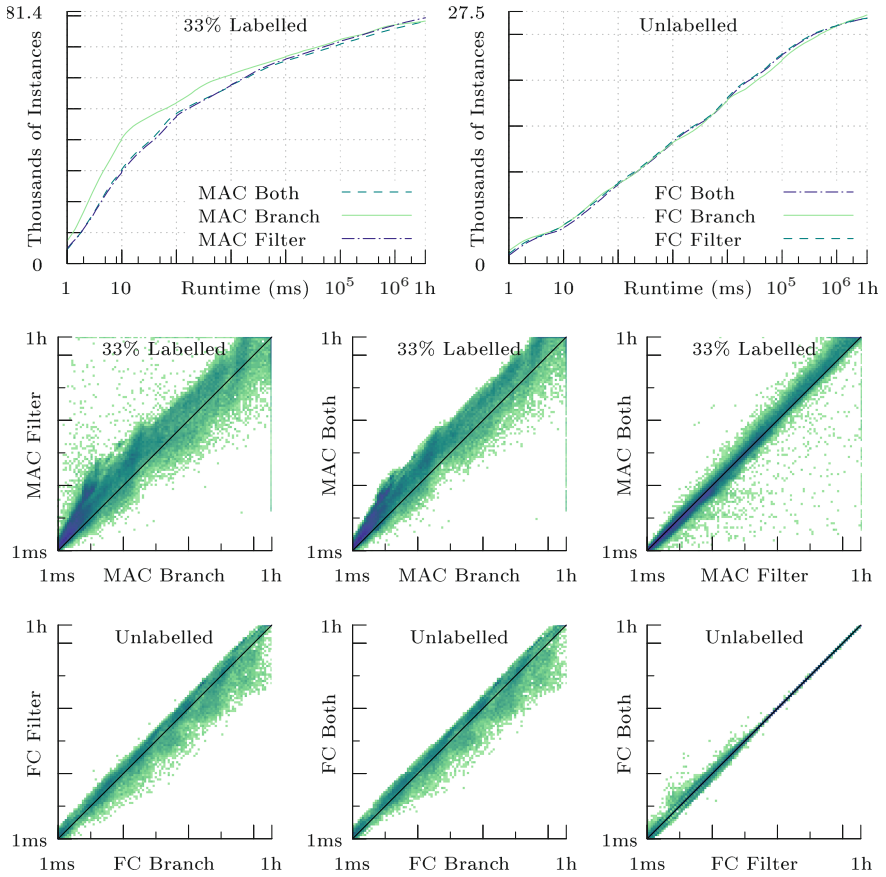
Fig. 5. On top, the cumulative number of MCCS instances solved in under a certain time using different CP techniques, for 33 % labelled (left) and unlabelled undirected (right) graphs. Below, instance-by-instance comparisons.

4.2 Experimental Comparison of CP Connectedness Techniques

Figure 5 compares the three approaches for ensuring connectedness in CP: by branching (Branch), by filtering (Filter), or by combining branching and filtering (Both). We show only results using the best variant for each class—that is, MAC for labelled graphs, and FC for unlabelled graphs (the other results are very similar). On labelled graphs, we see many instances which are solved very quickly by branching but not at all by filtering, and vice versa. However, combining both is rarely much worse than just doing one or the other, and is often much better, even if on average it is slightly slower. On unlabelled graphs, the three variants have rather similar performance.

4.3   Ensuring Connectedness in a Clique-Based Approach

It is not possible to determine connectedness from a raw association graph. However, we can take a maximum clique algorithm and mimic the CP branching strategy if we have access to the underlying graphs and can determine the "meaning" of the association graph vertices.

Most modern maximum clique algorithms for dense graphs use some variation of greedy graph colouring as a bound—the underlying observation is that each vertex in a clique must be given a different colour in a colouring, so if we can colour a subset of vertices using k colours then a maximum clique in this subset has at most k vertices. However, the colouring is also used as a branching heuristic: vertices are selected in reverse order from their colour classes in turn, starting with the last colour class created. Because of this coupling of branching and the bound (which is important in practice because it mimics a "smallest domain first" branching heuristic if colour classes are viewed as variables, without requiring a new bound to be calculated for every iteration [29]), if we were to select only a subset of vertices for branching at each stage inside a clique algorithm, we would lose completeness. Thus we must adapt the bound in a non-trivial way to take into account restricted branching.

In Algorithm 1 we present a novel clique-inspired algorithm which finds a maximum common connected induced subgraph via an association graph. If the additional branching restrictions are removed, the core of the algorithm is the same as the "MCSa1" clique algorithm used in the previous section (and we refer the reader to the previously cited works for implementation details on how to use bitsets and other data structures to implement the colouring stage with very low constant factors). The way we extend this algorithm for connectedness differs considerably from that of Koch [22] and Vismara and Valery [54]: these earlier approaches worked by classifying labels in the association graph based upon whether a common vertex is shared, and then constructing cliques with particular edge properties—this is harder to integrate with a strong bound function.

Our algorithm begins by building the association graph (line 4). The main part of the algorithm then works by building up candidate cliques in the *solution* variable, by recursive calls to the *search* procedure—starting from the empty set (line 5), each recursive subcall adds one vertex to *solution* (line 14) in such a way that *solution* is always a clique which corresponds to a connected common subgraph. The *remaining* set contains the set of vertices which are adjacent to every vertex in *solution*, and which have not yet been accepted or rejected (and so initially it contains every vertex). The main loops in the search procedure (lines 10 and 11) have the effect of iterating over each vertex in this set in a particular order—each vertex v is selected in turn, and then a recursive call is made to consider the effects of including v in *solution* (line 18), followed by the next iteration where v is instead rejected. When v is accepted, we add it to the new *solution′* (line 14), and create a new *remaining′* containing only the vertices in *remaining* which are adjacent to v (line 17).

The *connected* set contains the set of matching nodes which correspond to vertices adjacent to an already-accepted vertex in the first input

Algorithm 1. An algorithm for a maximum common connected induced subgraph isomorphism via an association graph.

```
1  associationMCCIS :: (Graph G₁, Graph G₂) → Map
2  begin
3  │    global incumbent ← ∅
4  │    G ← G₁ ▽ G₂
5  │    search(G, ∅, ∅, V(G))
6  │    return incumbent

7  search :: (Graph G, Set solution, Set connected, Set remaining)
8  begin
9  │    colourClasses ← concatenate(
   │        colour(G, remaining \ connected), colour(G, remaining ∩ connected))
10 │    while length(colourClasses) > 0 do
11 │    │    foreach v ∈ last(colourClasses) in reverse order do
12 │    │    │    if |solution| + length(colourClasses) ≤ |incumbent| then return
13 │    │    │    if v ∉ connected and solution ≠ ∅ then return
14 │    │    │    solution' ← solution ∪ {v}
15 │    │    │    if |solution'| > |incumbent| then incumbent ← solution'
16 │    │    │    connected' ← connected ∪ {w ∈ G : first(w) ∈ N(G, first(v))}
17 │    │    │    remaining' ← remaining ∩ N(G, v)
18 │    │    │    if remaining' ≠ ∅ then search(G, solution', connected', remaining')
19 │    │    removeLast(colourClasses)

20 colour :: (Graph G, Set uncoloured) → List of List of Vertex
21 begin
22 │    return a greedy colouring of the vertices in uncoloured, using the procedure
   │    of San Segundo et al. [47] with a static degree order from G and k_min = 0.
```

graph—in constraint programming terms, it is the set of assignments which could be made next which maintain connectedness. (Using only one of the two input graphs is sufficient for correctness, and has the advantage that the connectedness set may be determined by a simple lookup into a precomputed array which maps each vertex in the first input graph to a bitset.) At the top of search, this set is empty, and is not used (our first vertex selection is special, and does not care about connectedness). At subsequent depths, we may only accept vertices which are in this set, and if no such vertices remain then we return immediately (line 13). When recursing, we extend *connected* with the new vertices permitted by our acceptance of the branching v (line 16). Note that we are assuming that inside the main loops, we encounter every vertex in *remaining* ∩ *connected* before any vertex in *remaining* \ *connected*.

As we proceed, we keep track of the best solution we have found so far—this is stored in the *incumbent* variable (lines 3 and 15). We use the incumbent, together with a colour bound, to prune portions of the search space which cannot contain a better solution. The colour bound operates as follows: at each entry to the search procedure, we produce a greedy colouring of the vertices in *remaining* (line 9,
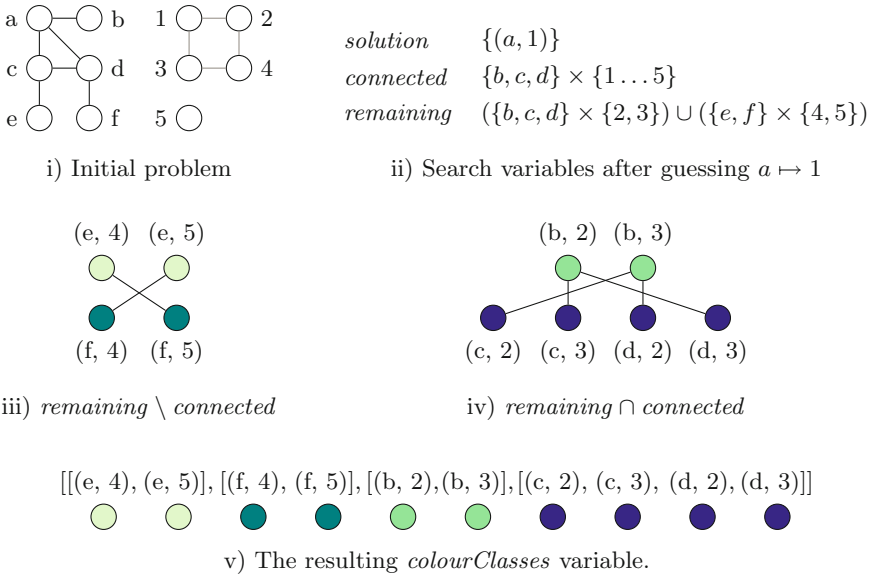
i) Initial problem

solution $\{(a, 1)\}$
connected $\{b, c, d\} \times \{1 \dots 5\}$
remaining $(\{b, c, d\} \times \{2, 3\}) \cup (\{e, f\} \times \{4, 5\})$

ii) Search variables after guessing $a \mapsto 1$

iii) *remaining* \ *connected*

iv) *remaining* ∩ *connected*

$[[(e, 4), (e, 5)], [(f, 4), (f, 5)], [(b, 2), (b, 3)], [(c, 2), (c, 3), (d, 2), (d, 3)]]$

v) The resulting *colourClasses* variable.

Fig. 6. Solving a maximum common connected problem using an association graph. Suppose we have already mapped vertex a to vertex 1, giving the assignments on the right. Now we have two subgraphs to colour. We need two colours for *remaining* \ *connected*, and we place these two colour classes first in the *colourClasses* variable. We can also colour *remaining* ∩ *connected* using two colours, since we cannot simultaneously map c to 2 and d to 3, or vice-versa. Thus *colourClasses* becomes a list of four colour classes. This tells us that if we hope to extend the current common subgraph by another four vertices, we must pick one assignment from each of the four colour classes (which is not actually possible, so the bound here gives an overestimate). The algorithm thus guesses $d \mapsto 3$ as its next assignment, and if that fails, $d \mapsto 2$, and so on; once $b \mapsto 3$ is reached, the bound decreases by one, and if $f \mapsto 5$ were reached we would stop due to a lack of remaining connected association nodes.

discussed further below). This greedy colouring gives us a list of colour classes, each of which is a list of pairwise non-adjacent vertices. The two loops (lines 10 and 11) then iterate over each colour class, from last to first, and then over each vertex in that colour class, again from last to first. (Rather than actually using a list of lists and removing items, this process should be implemented using a pair of immutable flat arrays. This technique is described elsewhere [29], so we do not discuss it here.) Finally, if at any point the number of remaining colour classes plus the number of vertices currently present in *solution* is not strictly greater than the size of the incumbent, then we may backtrack immediately (line 12).

Finally, we describe the colouring process—an example is shown in Fig. 6. In conventional clique algorithms, a simple greedy sequential colouring is used (possibly with the help of previous colourings to reduce the computational cost [35], and possibly with shortcuts taken for certain vertices [48], and possibly followed by a repair step to improve the colouring [53], or stronger bounding rules based
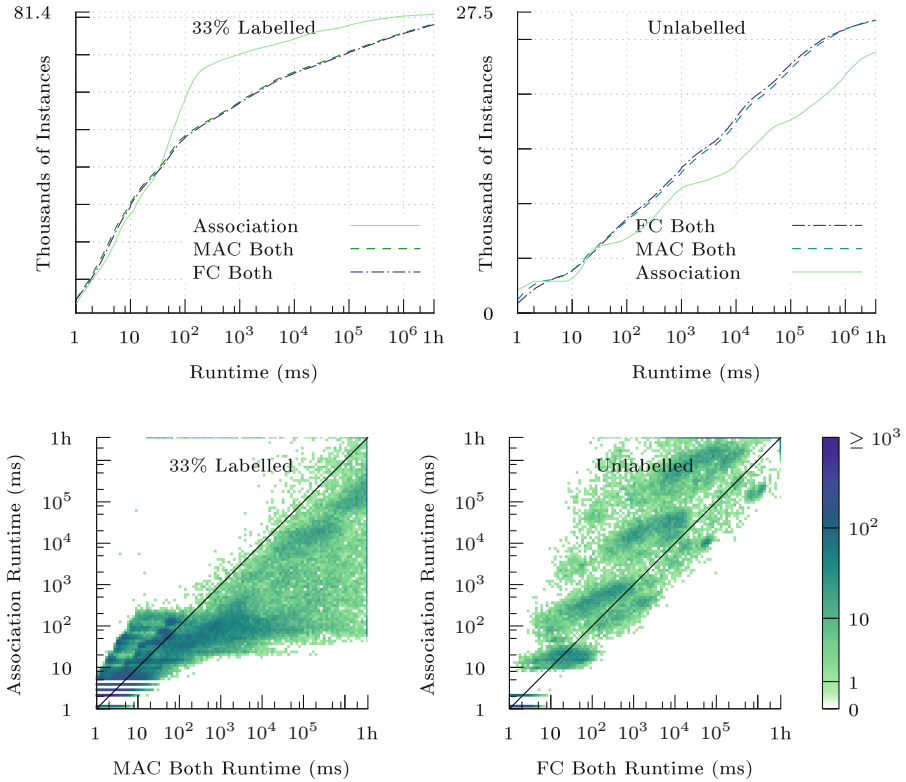
Fig. 7. The cumulative number of connected instances solved in under a certain time: on the top, 33 % labelled undirected graphs with up to 100 vertices, and then unlabelled and undirected graphs with up to 35 vertices. On the bottom, an instance-by-instance comparison of the association and CP Both approaches, with 33 % labelled graphs on the left, and unlabelled and undirected graphs on the right.

upon MaxSAT inference [25, 26, 46]). Such colourings will not give us the required property that vertices in *remaining* ∩ *connected* come last (so they are selected first by the reverse branching order). Thus we produce two greedy sequential colourings, first considering the non-branching vertices in *remaining* \ *connected*, followed by the branching vertices, and concatenate them (line 9). This produces a valid colouring, since we do not merge any colour classes between the two stages, although it may use more colours than a single colouring would.

(What if we did not guarantee that vertices in *remaining* ∩ *connected* came last, and just used a conventional colouring with the branching rule? Suppose we had four vertices in *remaining*, and produced a colouring $[[v_1, v_2], [v_3], [v_4]]$, and suppose that extending *solution* with $\{v_1, v_3, v_4\}$ gives an optimal solution. If v_4 was not *connected* yet, we would not branch on that subtree, and the bound could eliminate branching on v_3 and v_1, so we would miss the solution.

Thus we cannot simply add the branching rule without also adapting the combined bound and ordering heuristic.)

Our `colour` procedure is a simple greedy sequential colouring. We use the bit-parallel algorithm introduced by San Segundo *et al.* [47], with the k_{min} parameter set to 0, so we do not describe it here. We use a simple static degree ordering; other initial vertex orderings have been considered on general clique problems [38,43], and it is possible that special properties of the association graph could be exploited in this step (for example, it is always possible to colour the initial association graph using $\min(|V(G_1)|, |V(G_2)|)$ colours, but with certain vertex orderings, a greedy sequential colouring will sometimes use many more colours).

4.4   Experimental Comparison of the CP and Clique Approaches

In Fig. 7 we compare the clique-based approach to the connected problem with the two CP Both approaches. The trend is broadly similar to the unconnected problem: for labelled graphs, the clique-based approach is the clear winner, but for unlabelled graphs the clique approach lags somewhat.

The heatmaps show a more detailed picture. As before, in the unlabelled case, the association approach is almost never more than an order of magnitude better, and is often much worse. In the labelled case, however, there are now many instances where the CP approach does much better than the association approach, despite the association approach remaining much better overall.

5   Conclusion

Contradicting earlier claims in the literature, we have seen that a modern clique algorithm can perform competitively for maximum common subgraph problems, particularly when edge labels are involved. However, the best approaches for these problems is still far behind the state-of-the-art for subgraph isomorphism, where we can often scale to unlabelled graphs with thousands of vertices.

To start tackling this gap, we believe there is further scope for tailoring clique algorithms for association graphs, including specialised inference, a bound function which is aware that it is working on an association graph, and better initial vertex orderings. Treating the first branch specially may also be beneficial, since the first branch has an unusually large effect on the search space with association graphs [50].

For CP models, using a branching rule for connectedness, rather than simply as an ordering heuristic, is unconventional and does not cleanly fit into the abstractions used by toolkits. However, we saw that combining conventional filtering and the special branching rule was beneficial.

We looked only at single-threaded versions of these algorithms. Maximum clique algorithms have been extended for thread-parallel search [10,28,44], and in particular, work stealing strategies designed to eliminate exceptionally hard instances by forcing diversity at the top of search [30] could be beneficial in

eliminating some of the rare cases where the clique algorithm is many orders of magnitude worse than the CP models. On the CP side, the focus for parallelism has been on decomposition [33], rather than fully dynamic work stealing—it would be interesting to compare these approaches.

Finally, we intend to investigate larger and more diverse sets of instances, and other variants of the problem. We have yet to investigate partial or weighted graphs. Nor have we considered strongly connected common subgraphs—this would make the branching approach impossible, and filtering would be much more complicated. From the datasets we selected, there appears to be little scope for per-instance algorithm selection, but other families of input data could lead to a different conclusion.

References

1. Akutsu, T., Tamura, T.: A polynomial-time algorithm for computing the maximum common connected edge subgraph of outerplanar graphs of bounded degree. Algorithms **6**(1), 119–135 (2013). http://dx.doi.org/10.3390/a6010119
2. Balas, E., Yu, C.S.: Finding a maximum clique in an arbitrary graph. SIAM J. Comput. **15**(4), 1054–1068 (1986)
3. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16**(9), 575–577 (1957). http://doi.acm.org/10.1145/362342.362367
4. Brown, K.N., Prosser, P., Beck, C.J., Wu, C.W.: Exploring the use of constraint programming for enforcing connectivity during graph generation. In: Proceedings IJCAI Workshop on Modelling and Solving Problems with Constraints, Edinburgh, Scotland, pp. 26–31 (2005)
5. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recogn. Lett. **18**(8), 689–694 (1997). http://dx.doi.org/10.1016/S0167-8655(97)00060-3
6. Bunke, H., Foggia, P., Guidobaldi, C., Sansone, C., Vento, M.: A comparison of algorithms for maximum common subgraph on randomly connected graphs. In: Caelli, T.M., Amin, A., Duin, R.P.W., Kamel, M.S., de Ridder, D. (eds.) SPR 2002 and SSPR 2002. LNCS, vol. 2396, pp. 123–132. Springer, Heidelberg (2002). http://dx.doi.org/10.1007/3-540-70659-3_12
7. Combier, C., Damiand, G., Solnon, C.: Map edit distance vs. graph edit distance for matching images. In: Kropatsch, W.G., Artner, N.M., Haxhimusa, Y., Jiang, X. (eds.) GbRPR 2013. LNCS, vol. 7877, pp. 152–161. Springer, Heidelberg (2013)
8. Conte, D., Foggia, P., Vento, M.: Challenging complexity of maximum common subgraph detection algorithms: a performance analysis of three algorithms on a wide database of graphs. J. Graph Algorithms Appl. **11**(1), 99–143 (2007). http://jgaa.info/accepted/2007/ConteFoggiaVento2007.11.1.pdf
9. Cook, D.J., Holder, L.B.: Substructure discovery using minimum description length and background knowledge. J. Artif. Intell. Res. (JAIR) **1**, 231–255 (1994). http://dx.doi.org/10.1613/jair.43
10. Depolli, M., Konc, J., Rozman, K., Trobec, R., Janezic, D.: Exact parallel maximum clique algorithm for general and protein graphs. J. Chem. Inf. Model. **53**(9), 2217–2228 (2013). http://dx.doi.org/10.1021/ci4002525

11. Dhaenens, C., Jourdan, L., Marmion, M. (eds.): Learning and Intelligent Optimization. LNCS, vol. 8994. Springer, Switzerland (2015). http://dx.doi.org/10.1007/978-3-319-19084-6

12. Djoko, S., Cook, D.J., Holder, L.B.: An empirical study of domain knowledge and its benefits to substructure discovery. IEEE Trans. Knowl. Data Eng. **9**(4), 575–586 (1997). http://dx.doi.org/10.1109/69.617051

13. Dooms, G., Deville, Y., Dupont, P.E.: CP(Graph): introducing a graph computation domain in constraint programming. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 211–225. Springer, Heidelberg (2005). http://dx.doi.org/10.1007/11564751_18

14. Droschinsky, A., Kriege, N., Mutzel, P.: Faster algorithms for the maximum common subtree isomorphism problem. In: Faliszewski, P., Muscholl, A., Niedermeier, R. (eds.) 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016). Leibniz International Proceedings in Informatics (LIPIcs), vol. 58, pp. 34:1–34:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2016, to appear)

15. Durand, P.J., Pasari, R., Baker, J.W., Tsai, C.C.: An efficient algorithm for similarity analysis of molecules. Internet J. Chem. **2**(17), 1–16 (1999)

16. Ehrlich, H.C., Rarey, M.: Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. Wiley Interdisc. Rev. Comput. Mol. Sci. **1**(1), 68–79 (2011). http://dx.doi.org/10.1002/wcms.5

17. Englert, P., Kovács, P.: Efficient heuristics for maximum common substructure search. J. Chem. Inf. Model. **55**(5), 941–955 (2015). http://dx.doi.org/10.1021/acs.jcim.5b00036

18. Fernández, M., Valiente, G.: A graph distance metric combining maximum common subgraph and minimum common supergraph. Pattern Recogn. Lett. **22**(6/7), 753–758 (2001)

19. Gao, D., Reiter, M.K., Song, D.: BinHunt: automatically finding semantic differences in binary programs. In: Chen, L., Ryan, M.D., Wang, G. (eds.) ICICS 2008. LNCS, vol. 5308, pp. 238–255. Springer, Heidelberg (2008). http://dx.doi.org/10.1007/978-3-540-88625-9_16

20. Gay, S., Fages, F., Martinez, T., Soliman, S., Solnon, C.: On the subgraph epimorphism problem. Discrete Appl. Math. **162**, 214–228 (2014). http://dx.doi.org/10.1016/j.dam.2013.08.008

21. Jégou, P.: Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In: Fikes, R., Lehnert, W.G. (eds.) Proceedings of the 11th National Conference on Artificial Intelligence, Washington, DC, USA, pp. 731–736. AAAI Press/The MIT Press, 11–15 July 1993. http://www.aaai.org/Library/AAAI/1993/aaai93-109.php

22. Koch, I.: Enumerating all connected maximal common subgraphs in two graphs. Theor. Comput. Sci. **250**(1–2), 1–30 (2001). http://dx.doi.org/10.1016/S0304-3975(00)00286-3

23. Kriege, N.: Comparing graphs. Ph.d. thesis, Technische Universität Dortmund (2015)

24. Levi, G.: A note on the derivation of maximal common subgraphs of two directed or undirected graphs. CALCOLO **9**(4), 341–352 (1973). http://dx.doi.org/10.1007/BF02575586

25. Li, C., Fang, Z., Xu, K.: Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, pp. 939–946. IEEE Computer Society, 4–6 November 2013. http://dx.doi.org/10.1109/ICTAI.2013.143

26. Li, C., Jiang, H., Xu, R.: Incremental MaxSAT reasoning to reduce branches in a branch-and-bound algorithm for MaxClique. In: Dhaenens et al. [11], pp. 268–274. http://dx.doi.org/10.1007/978-3-319-19084-6_26

27. Lu, S.W., Ren, Y., Suen, C.Y.: Hierarchical attributed graph representation and recognition of handwritten chinese characters. Pattern Recogn. **24**(7), 617–632 (1991). http://www.sciencedirect.com/science/article/pii/0031320391900295

28. McCreesh, C., Prosser, P.: Multi-threading a state-of-the-art maximum clique algorithm. Algorithms **6**(4), 618–635 (2013). http://dx.doi.org/10.3390/a6040618

29. McCreesh, C., Prosser, P.: Reducing the branching in a branch and bound algorithm for the maximum clique problem. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 549–563. Springer, Heidelberg (2014). http://dx.doi.org/10.1007/978-3-319-10428-7_40

30. McCreesh, C., Prosser, P.: The shape of the search tree for the maximum clique problem and the implications for parallel branch and bound. TOPC **2**(1), 8 (2015). http://doi.acm.org/10.1145/2742359

31. McCreesh, C., Prosser, P., Trimble, J.: Heuristics and really hard instances for subgraph isomorphism problems. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence (2016, to appear)

32. McGregor, J.J.: Backtrack search algorithms and the maximal common subgraph problem. Softw. Pract. Exp. **12**(1), 23–34 (1982)

33. Minot, M., Ndiaye, S.N., Solnon, C.: A comparison of decomposition methods for the maximum common subgraph problem. In: 27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, pp. 461–468. IEEE, 9–11 November 2015. http://dx.doi.org/10.1109/ICTAI.2015.75

34. Ndiaye, S.N., Solnon, C.: CP models for maximum common subgraph problems. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 637–644. Springer, Heidelberg (2011). http://dx.doi.org/10.1007/978-3-642-23786-7_48

35. Nikolaev, A., Batsyn, M., Segundo, P.S.: Reusing the same coloring in the child nodes of the search tree for the maximum clique problem. In: Dhaenens et al. [11], pp. 275–280. http://dx.doi.org/10.1007/978-3-319-19084-6_27

36. Petit, T., Régin, J.-C., Bessière, C.: Specific filtering algorithms for over-constrained problems. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 451–463. Springer, Heidelberg (2001). http://dx.doi.org/10.1007/s10479-011-1019-8

37. Piva, B., de Souza, C.C.: Polyhedral study of the maximum common induced subgraph problem. Ann. OR **199**(1), 77–102 (2012). http://dx.doi.org/10.1007/s10479-011-1019-8

38. Prosser, P.: Exact algorithms for maximum clique: a computational study. Algorithms **5**(4), 545–587 (2012). http://dx.doi.org/10.3390/a5040545

39. Prosser, P., Unsworth, C.: A connectivity constraint using bridges. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) Proceedings of the 17th European Conference on Artificial Intelligence, ECAI 2006. Frontiers in Artificial Intelligence and Applications, vol. 141, August 29–September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS), pp. 707–708. IOS Press (2006)

40. Raymond, J.W., Willett, P.: Maximum common subgraph isomorphism algorithms for the matching of chemical structures. J. Comput. Aided Mol. Des. **16**(7), 521–533 (2002). http://dx.doi.org/10.1023/A:1021271615909

41. Sabin, D., Freuder, E.C.: Contradicting conventional wisdom in constraint satisfaction. In: Borning, A. (ed.) PPCP 1994. LNCS, vol. 874, pp. 10–20. Springer, Heidelberg (1994)

42. Santo, M.D., Foggia, P., Sansone, C., Vento, M.: A large database of graphs and its use for benchmarking graph isomorphism algorithms. Pattern Recogn. Lett. **24**(8), 1067–1079 (2003). http://dx.doi.org/10.1016/S0167-8655(02)00253-2

43. Segundo, P.S., Lopez, A., Batsyn, M.: Initial sorting of vertices in the maximum clique problem reviewed. In: Pardalos, P.M., Resende, M.G.C., Vogiatzis, C., Walteros, J.L. (eds.) LION 2014. LNCS, vol. 8426, pp. 111–120. Springer, Switzerland (2014). http://dx.doi.org/10.1007/978-3-319-09584-4_12

44. Segundo, P.S., Lopez, A., Pardalos, P.M.: A new exact maximum clique algorithm for large and massive sparse graphs. Comput. OR **66**, 81–94 (2016). http://dx.doi.org/10.1016/j.cor.2015.07.013

45. Segundo, P.S., Matía, F., Rodríguez-Losada, D., Hernando, M.: An improved bit parallel exact maximum clique algorithm. Optim. Lett. **7**(3), 467–479 (2013). http://dx.doi.org/10.1007/s11590-011-0431-y

46. Segundo, P.S., Nikolaev, A., Batsyn, M.: Infra-chromatic bound for exact maximum clique search. Comput. OR **64**, 293–303 (2015). http://dx.doi.org/10.1016/j.cor.2015.06.009

47. Segundo, P.S., Rodríguez-Losada, D., Jiménez, A.: An exact bit-parallel algorithm for the maximum clique problem. Comput. OR **38**(2), 571–581 (2011). http://dx.doi.org/10.1016/j.cor.2010.07.019

48. Segundo, P.S., Tapia, C.: Relaxed approximate coloring in exact maximum clique search. Comput. OR **44**, 185–192 (2014). http://dx.doi.org/10.1016/j.cor.2013.10.018

49. Shasha, D., Wang, J.T.L., Giugno, R.: Algorithmics and applications of tree and graph searching. In: Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2002, NY, USA, pp. 39–52 (2002). http://doi.acm.org/10.1145/543613.543620

50. Suters, W.H., Abu-Khzam, F.N., Zhang, Y., Symons, C.T., Samatova, N.F., Langston, M.A.: A new approach and faster exact methods for the maximum common subgraph problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 717–727. Springer, Heidelberg (2005). http://dx.doi.org/10.1007/11533719_73

51. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Global Optim. **37**(1), 95–111 (2007). http://dx.doi.org/10.1007/s10898-006-9039-7

52. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731. Springer, Heidelberg (2003). http://dx.doi.org/10.1007/3-540-45066-1_22

53. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: Rahman, M.S., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 191–203. Springer, Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-11440-3_18

54. Vismara, P., Valery, B.: Finding maximum common connected subgraphs using clique detection or constraint satisfaction algorithms. In: An, L.T.H., Bouvry, P., Tao, P.D. (eds.) MCO 2008. CCIS, vol. 14, pp. 358–368. Springer, Heidelberg (2008). http://dx.doi.org/10.1007/978-3-540-87477-5_39

Tightening McCormick Relaxations for Nonlinear Programs via Dynamic Multivariate Partitioning

Harsha Nagarajan[1]([⊠]), Mowen Lu[2], Emre Yamangil[1], and Russell Bent[1]

[1] Center for Nonlinear Studies, Los Alamos National Laboratory,
Los Alamos, NM, USA
{harsha,emreyamangil,rbent}@lanl.gov
[2] Department of Industrial Engineering, Clemson University, Clemson, SC, USA
mlu87@g.clemson.edu

Abstract. In this work, we propose a two-stage approach to strengthen piecewise McCormick relaxations for mixed-integer nonlinear programs (MINLP) with multi-linear terms. In the first stage, we exploit Constraint Programing (CP) techniques to contract the variable bounds. In the second stage we partition the variables domains using a dynamic multivariate partitioning scheme. Instead of equally partitioning the domains of variables appearing in multi-linear terms, we construct sparser partitions yet tighter relaxations by iteratively partitioning the variable domains in regions of interest. This approach decouples the number of partitions from the size of the variable domains, leads to a significant reduction in computation time, and limits the number of binary variables that are introduced by the partitioning. We demonstrate the performance of our algorithm on well-known benchmark problems from MINLPLIB and discuss the computational benefits of CP-based bound tightening procedures.

Keywords: McCormick relaxations · MINLP · Dynamic partitioning · Bound tightening

1 Introduction

Mixed Integer Nonlinear Programs (MINLPs) are part of a class of non-convex, mathematical programs that include discrete variables and nonlinear terms in the objective function and/or constraints. Within many application domains, MINLPs with multi-linear, non-convex terms are of great interest. For example, these problems appear in chemical engineering (synthesis of process/water networks) [18,20], energy infrastructure networks [8], and in the molecular distance geometry problem [16]. Despite their importance in such areas, these problems remain difficult to solve. Global optimization solvers, like BARON [21], depend heavily on the quality of mixed-integer linear programing relaxations to MINLPs. However, these relaxations are often weak and the solvers are not guaranteed to

converge to a global optimum or even find a feasible solution. As a result, there is considerable interest in developing tighter relaxations that improve the convergence of global solvers. In this paper we focus on MINLPs with multi-linear terms, though the approach is generalizable.

In the context of this paper, there are two key methods for deriving tight relaxations of MINLPs with multi-linear terms. First, variable bounds are a critical contributor to the quality of relaxations. As a result, bound tightening methods have received a great deal of attention, in particular for problems with bilinear terms [1,5,6,9,19]. In most of these papers, the most common approaches solve sequences of minimization and maximization problems where the continuous variables are the objective. The solutions to these problems are used to tighten the bounds of the variables. In this paper, we combine these bound tightening approaches with constraint programming to improve their effectiveness. The second method for tightening relaxations focuses on reducing the size of the relaxed feasible space. This is often done with domain partitioning, i.e. spatial branch-and-bound (sBB). In sBB, a single variable (often the variable that violates the feasible region the most) is iteratively partitioned within a branch-and-bound search tree [22,23]. One of the crucial requirements of successful sBB is tight lower bounds on the objective. These bounds support efficient pruning of infeasible regions and some of the most effective bounds are those that are based on relaxations. When multi-linear terms are involved, a commonly used method is McCormick relaxations. As McCormick relaxations tend to be loose in many situations, the literature contains many efforts to improve these relaxations. The method most closely related to our own builds uniform piecewise McCormick relaxations via univariate or bivariate partitioning [6,11,14]. One of the drawbacks of such approaches is that they may need a large number of partitions that are controlled by on/off binary variables. As these binary variables introduce combinatorial inefficiencies, this approach is often restricted to small problems. To address this issue, there has been recent work focusing on addressing these inefficiencies. For example, [5,7] combines multiparametric disaggregation with optimality-based bound tightening methods. In [25], the authors discuss a non-uniform, bivariate partitioning approach that improves relaxations but provide results for a single, simple benchmark problem. More recently, in [11], the authors report the advantages of bivariate (as compared to univariate) partitioning, however they use partitions chosen at uniformly located grid points. In the context of multi-linear terms, [24] discusses a univariate parametrization method that solves medium-sized benchmarks. However, none of these approaches address the key limitation of uniform partitioning, *partition density*, i.e. these methods introduce partitions in unproductive regions of the search space. We address this limitation by introducing an approach that dynamically partitions the relaxations in regions of the search space that favor optimality. To the best of our knowledge, there is little or no work on methods for solving MINLPs with multi-linear terms with such sparse partitioning.

To summarize, we address the problem of tight relaxations for non-convex multi-linear functions and we develop a two-stage algorithm that strengthens

piecewise multi-linear relaxations. In the first stage, we apply a sequential, CP inspired, bound tightening approach. In the second stage, we develop a dynamic, sparse multivariate partitioning approach that addresses the key limitations of uniform partitioning approaches. With this algorithm, we are able to solve many MINLPs more efficiently and accurately with fewer parameter tuning options than the existing approaches. The remainder of this paper is organized as follows: Sect. 2 discusses the required notation, problem set up and reviews McCormick relaxations for bilinear and multi-linear terms. Section 3 discusses a sequential bound tightening approach, formalizes the concepts and notation for piecewise relaxations of McCormick envelopes, and provides a detailed discussion on multivariate dynamic partitioning algorithm on multi-linear and monomial terms. Section 4 illustrates the strength of the proposed algorithms on benchmark MINLPs and Sect. 5 concludes the paper.

2 Problem Definition

Notation: Here, we use lower and upper case for vector and matrix entries, respectively. Bold font refers to the entire vector or matrix. With this notation, $||\mathbf{v}||_2$ defines the L_2 norm of vector $\mathbf{v} \in \mathbb{R}^n$. Given vectors $\mathbf{v}_1 \in \mathbb{R}^n$ and $\mathbf{v}_2 \in \mathbb{R}^n$, $\mathbf{v}_1 \cdot \mathbf{v}_2 = \sum_{i=1}^{n} v_{1i}v_{2i}$; $\mathbf{v}_1 + \mathbf{v}_2$ implies element-wise sums; and $\frac{\mathbf{v}_1}{\alpha}$ denotes the element-wise ratio between entries of $\mathbf{v}_1$ and the scalar α. Next, $z \in \mathbb{Z}^+$ represents a strictly positive integer scalar. $\mathbf{M} \in \mathbb{S}^{n \times n}$ represents a symmetric square matrix $\mathbf{M}$. Given variables x_i and x_j, $\langle x_i, x_j \rangle^{MC}$, $\langle x_i, x_j \rangle^{UTMC}$ and $\langle x_i, x_j \rangle^{DTMC}$ denote the McCormick envelope, uniformly-partitioned McCormick envelope and dynamically-partitioned McCormick envelope, respectively. (x_i^L, x_i^U) denotes the prescribed global lower and upper bound and (x_i^l, x_i^u) denotes the tightened lower and upper bound, respectively.

Problem: The problems considered in this paper are MINLPs, where the non-linearity is due to multi-linear (polynomial) functions. Often, these problems are not convex. The general form of the problem, denoted as $\mathcal{P}_0$, is as follows:

$$\begin{aligned}
\mathcal{P}_0: \quad &\underset{\mathbf{x},\mathbf{y}}{\text{minimize}} \quad f(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\
&\text{subject to} \ \ g(\mathbf{x}, \mathbf{y}, \mathbf{z}) \leq 0, \\
&\qquad\qquad h(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0, \\
&\qquad\qquad z_K = x_i x_j \ldots x_k, \quad \forall K \in ML \\
&\qquad\qquad \mathbf{x}^L \leq \mathbf{x} \leq \mathbf{x}^U, \\
&\qquad\qquad \mathbf{y} \in \{0,1\}^m
\end{aligned}$$

where, $f : \mathbb{R}^n \to \mathbb{R}$ is a scalar multi-linear function and $g : \mathbb{R}^n \to \mathbb{R}^{m_1}, h : \mathbb{R}^n \to \mathbb{R}^{m_2}$ are vector, multi-linear functions. $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are vectors of continuous variables with box constraints, binary variables, and multi-linear functions, respectively. z_K is the Kth multilinear term in the set ML such that $ML = \{K = (i, j, \ldots, k)|z_K = x_i x_j \ldots x_k\}$. When $i = j = \ldots = k$, the multi-linearity is reduced to monomial terms.

2.1 Standard Convex Relaxations for Multi-linear Terms

McCormick relaxations: Given two variables, $x_i, x_j \in \mathbb{R}$ such that $x_i^l \leq x_i \leq x_i^u$ and $x_j^l \leq x_j \leq x_j^u$, we define the McCormick relaxation [17] of the bilinear product $x_i x_j$ as $\widehat{x_{ij}} \in \langle x_i, x_j \rangle^{MC}$ such that $\widehat{x_{ij}}$ satisfies

$$\widehat{x_{ij}} \geq x_i^l x_j + x_j^l x_i - x_i^l x_j^l \tag{1a}$$

$$\widehat{x_{ij}} \geq x_i^u x_j + x_j^u x_i - x_i^u x_j^u \tag{1b}$$

$$\widehat{x_{ij}} \leq x_i^l x_j + x_j^u x_i - x_i^l x_j^u \tag{1c}$$

$$\widehat{x_{ij}} \leq x_i^u x_j + x_j^l x_i - x_i^u x_j^l \tag{1d}$$

The relaxations in (1) are exact when one of the variables involved in the product is a binary variable. Further, relaxations in (1) can be reduced to a simpler form (three constraints) when both the variables involved in the product are binary variables. If y_i and y_j are binary variables, we denote this simplified relaxation as $\widehat{y_{ij}} \in \langle y_i, y_j \rangle^{BMC}$.

Successive McCormick relaxations of multi-linear terms: Given a multi-linear term $x_i x_j \ldots x_k$ with k-linear terms, we use a general technique for successively deriving McCormick envelopes on bilinear combinations of the terms. As discussed in [4], the tightness of McCormick relaxations depends on the grouping order of bilinear terms. Here, we assume a lexicographic order of grouping the bilinear terms. For example, given a multi-linear term $(x_1 x_2 x_3 x_4)$, the successive ordering of bilinear terms is $(((x_1 x_2) x_3) x_4)$. More formally, for k-linear terms, the McCormick envelope of $x_i x_j \ldots x_{k-1} x_k$ is represented as

$$\langle x_i x_j \ldots x_{k-1} x_k \rangle^{MC} = \langle \langle \langle x_i x_j \rangle^{MC} \ldots x_{k-1} \rangle^{MC} x_k \rangle^{MC}.$$

Study of alternate grouping choices is beyond the scope of this paper and is a topic of future work.

3 CP-DTMC Algorithm

The Constraint Programming with Dynamic Tightening of McCormicks (CP-DTMC) algorithm is described in this section. It combines CP based domain tightening with a partitioning scheme for McCormick relaxations.

3.1 Sequential Bound Tightening Procedure

The first stage of CP-DTMC tightens the bounds of the continuous variables of $\mathcal{P}_0$. In many engineering applications there is little or no information about the upper and lower bounds $(\mathbf{x}^L, \mathbf{x}^U)$ of these variables. Even when known, the gap between the bounds is often large. As discussed earlier, these bounds are used in McCormick relaxations to derive convex envelopes of multi-linear terms in $\mathcal{P}_0$. Large bounds generally weaken these relaxations, degrade the quality of the lower bounds, and slow the convergence of branch-and-cut algorithms.

In practice, replacing the original bounds with tighter bounds can (sometimes) dramatically improve the quality of these relaxations (see Fig. 1[a]).

The basic idea of bound tightening is to derive (new) valid bounds to improve the relaxations. Our approach is based on the work [6] and is related to the iterative bound tightening of [8]. Let $x_i, i = 1, \ldots, n$ be the element-wise entries of a continuous variable vector $\mathbf{x} \in \mathbb{R}^n$. In order to shrink the bounds of x_i, we solve a modified version of $\mathcal{P}_0$. For each x_i, we first solve $\mathcal{P}_0$ where we minimize x_i and then solve $\mathcal{P}_0$ where we maximize x_i. In both cases we add a constraint that bounds the original objective function of $\mathcal{P}_0$ with a best known feasible solution $(\mathbf{x}_{loc}^*, \mathbf{y}_{loc}^*, \mathbf{z}_{loc}^*)$. This is a key difference between our approach and [6,8]. We also iteratively tighten the domain (bounds) of the variables using the approach above. While there are other CP propagation methods that could be used to further improve the quality of the bounds, this method was sufficient to demonstrate the effectiveness of the overall approach.

More formally, Algorithm 1 describes the first stage of CP-DTMC. Line 1 takes as input the current bounds and a feasible solution. The core of the algorithm is embedded in Line 4. This is where we solve the variations of $\mathcal{P}_0$. Line 4a states the minimization and maximization of x_i. Line 4b adds a bound on the original objective function. Lines 4c–4f state the rest of $\mathcal{P}_0$. Based on these solutions, we update the bounds of our variables (line 5). The procedure continues until the bounds do not change (line 2). Algorithm 1 is naturally parallel as each MILP of line 4 is independently solvable.

Algorithm 1. Sequential bound tightening on $\mathbf{x}$ vector

---

1: Input: $\mathbf{x}^l \leftarrow \mathbf{x}^L, \mathbf{x}^u \leftarrow \mathbf{x}^U, \mathbf{x}_{iter}^l = \mathbf{x}_{iter}^u \leftarrow \mathbf{0}, \mathbf{x}_{loc}^*, \mathbf{y}_{loc}^*, \mathbf{z}_{loc}^*, TOL > 0$.

2: **while** $||\mathbf{x}^l - \mathbf{x}_{iter}^l||_2 > TOL$ and $||\mathbf{x}^u - \mathbf{x}_{iter}^u||_2 > TOL$ **do**

3: $\quad \mathbf{x}_{iter}^l \leftarrow \mathbf{x}^l, \; \mathbf{x}_{iter}^u \leftarrow \mathbf{x}^u$

4: $\quad$ Solve:

$$x_i^{*l} := \min_{\mathbf{x},\mathbf{y}} x_i; \quad x_i^{*u} := \max_{\mathbf{x},\mathbf{y}} x_i \; \forall i = 1, \ldots, n \qquad (a)$$

$$\text{subject to} \quad f(\mathbf{x},\mathbf{y},\mathbf{z}) \leq f(\mathbf{x}_{loc}^*, \mathbf{y}_{loc}^*, \mathbf{z}_{loc}^*), \qquad (b)$$

$$g(\mathbf{x},\mathbf{y},\mathbf{z}) \leq 0, \qquad (c)$$

$$h(\mathbf{x},\mathbf{y},\mathbf{z}) = 0, \qquad (d)$$

$$z_K = \langle x_i x_j \ldots x_k \rangle^{MC}, \quad \forall K \in ML \quad (d)$$

$$\mathbf{x}_{iter}^l \leq \mathbf{x} \leq \mathbf{x}_{iter}^u, \qquad (e)$$

$$\mathbf{y} \in \{0,1\}^m \qquad (f)$$

5: $\quad \mathbf{x}^l \leftarrow \mathbf{x}^{*l}, \; \mathbf{x}^u \leftarrow \mathbf{x}^{*u}$

6: **end while**

7: return $\mathbf{x}^l, \mathbf{x}^u$ (tightened bounds).

3.2   Algorithm for Global Optimization of MINLPs

The second stage of CP-DTMC derives piecewise McCormick relaxations of multi-linear terms based on multivariate dynamic partitioning. In practice, partitioning the bounds of the variables of the McCormick tightens the overall relaxation. As the number of partitions goes to ∞, partitioning exactly approximates the original multi-linear terms. However, introducing a large number of partitions generally renders the problem intractable because the choice of partition is controlled by binary on/off variables. Thus, typical approaches assume a (small) finite number of partitions that uniformly discretize the multi-linear variables [2,6,10,11]. While this is a straight-forward method for partitioning the domain of variables, it potentially creates partitions that correspond to solutions that are far away from the optimality region of the search space. In other words, many of the partitions are not useful. Instead, we develop an approach that successively tightens the McCormick relaxations with sparse domain discretization. This approach focuses partitioning on areas of the variable domain that appear to influence optimality the most.

Lower bounds using piecewise McCormick relaxations: Without loss of generality and for ease of explanation, we restrict the discussion of the lower bounding procedure to bilinear terms[1]. Given a bilinear term $x_i x_j$, we partition the domains of x_i and x_j into $M_i \in \mathbb{Z}^+$ and $M_j \in \mathbb{Z}^+$ disjoint regions with new binary variables $\hat{\mathbf{y}}_i \in \{0,1\}^{M_i}$ and $\hat{\mathbf{y}}_j \in \{0,1\}^{M_j}$ added to the formulation. The binary variables are used to control the partitions that are active and the tighter relaxation associated with the active partition. Formally, the piecewise McCormick constraints for a bilinear term, denoted by $\widehat{x_{ij}} \in \langle x_i, x_j \rangle^{UTMC}$ (uniform partitioning) or $\widehat{x_{ij}} \in \langle x_i, x_j \rangle^{DTMC}$ (dynamic partitioning), take the following form:

$$\widehat{x_{ij}} \geq (\mathbf{x}_i^l \cdot \hat{\mathbf{y}}_i)x_j + (\mathbf{x}_j^l \cdot \hat{\mathbf{y}}_j)x_i - (\mathbf{x}_i^l \cdot \hat{\mathbf{y}}_i)(\mathbf{x}_j^l \cdot \hat{\mathbf{y}}_j) \tag{2a}$$

$$\widehat{x_{ij}} \geq (\mathbf{x}_i^u \cdot \hat{\mathbf{y}}_i)x_j + (\mathbf{x}_j^u \cdot \hat{\mathbf{y}}_j)x_i - (\mathbf{x}_i^u \cdot \hat{\mathbf{y}}_i)(\mathbf{x}_j^u \cdot \hat{\mathbf{y}}_j) \tag{2b}$$

$$\widehat{x_{ij}} \leq (\mathbf{x}_i^l \cdot \hat{\mathbf{y}}_i)x_j + (\mathbf{x}_j^u \cdot \hat{\mathbf{y}}_j)x_i - (\mathbf{x}_i^l \cdot \hat{\mathbf{y}}_i)(\mathbf{x}_j^u \cdot \hat{\mathbf{y}}_j) \tag{2c}$$

$$\widehat{x_{ij}} \leq (\mathbf{x}_i^u \cdot \hat{\mathbf{y}}_i)x_j + (\mathbf{x}_j^l \cdot \hat{\mathbf{y}}_j)x_i - (\mathbf{x}_i^u \cdot \hat{\mathbf{y}}_i)(\mathbf{x}_j^l \cdot \hat{\mathbf{y}}_j) \tag{2d}$$

$$\sum_{k=1}^{M_i} \hat{y}_{i_k} = 1, \quad \sum_{k=1}^{M_j} \hat{y}_{j_k} = 1 \tag{2e}$$

$$\hat{\mathbf{y}}_i \in \{0,1\}^{M_i}, \hat{\mathbf{y}}_j \in \{0,1\}^{M_j} \tag{2f}$$

where, $(\mathbf{x}_i^l, \mathbf{x}_i^u) \in \mathbb{R}^{M_i}$ are the lower and upper bound vectors of variable x_i for each partition. In other words, for the kth partition of x_i, the following constraint defines the partition: $x_{i_k}^l \leq x_i \leq x_{i_k}^u$. Note that the bilinear terms in $\hat{\mathbf{y}}_j x_i$ and $\hat{\mathbf{y}}_i x_j$ are exactly linearized using standard McCormick relaxations. Also, $(\mathbf{x}_i^l \cdot \hat{\mathbf{y}}_i)(\mathbf{x}_j^l \cdot \hat{\mathbf{y}}_j)$ is rewritten as $\mathbf{x}_i^l(\hat{\mathbf{y}}_i\hat{\mathbf{y}}_j^T)\mathbf{x}_j^l$, where $\hat{\mathbf{Y}} = (\hat{\mathbf{y}}_i\hat{\mathbf{y}}_j^T)$ is an $M_i \times M_j$

---

[1] This approach is easily extended to multi-linear terms using successive bilinear relaxations as discussed in Sect. 2.1.

(a) Bilinear term $(x_i x_j)$ (b) Monomial term (x_i^2)
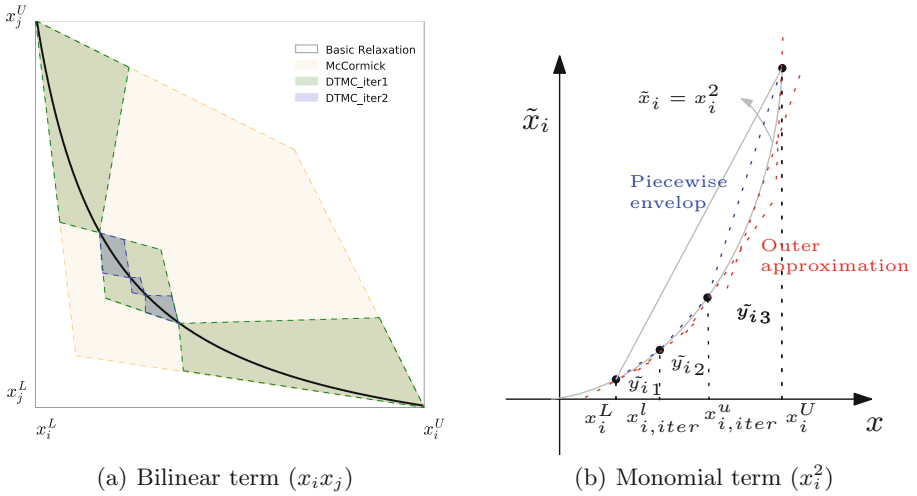
Fig. 1. Feasible regions for bilinear and monomial (quadratic) terms based on DTMC.

matrix with binary product entries. As discussed in Sect. 2.1, any binary product entry, $y_i y_j$, of $\hat{\mathbf{Y}}$ is exactly represented as $\langle y_i, y_j \rangle^{BMC}$.

CP-DTMC algorithm for multi-linear terms. Given this model of piecewise McCormick relaxations, we can now formalize dynamically tightening of these relaxations. The pseudo-code of the DTMC algorithm is outlined in Algorithm 2. The full CP-DTMC algorithm combines Algorithm 1 with Algorithm 2 and is described in Algorithm 3. We first discuss the dynamic partitioning scheme as outlined in Algorithm 2 followed by the discussion of Algorithm 3.

We first define $\boldsymbol{P}_{iter}^*$ as a vector of active partitions whose dimension is equal to $|\mathbf{x}|$. For any variable x_i, an active partition contains a lower bound and an upper bound for x_i The choice of the active partition of x_i is the binary variable of vector $\hat{\mathbf{y}}_i$ whose component is equal to 1.0. As shown in line 3 of Algorithm 2, the size of the partition is dependent on the size of the active partition of the current solution $\mathbf{x}_{iter}^*$. The parameter, Δ, is used to scale the partition's size and it influences the convergence speed and the number of partitions. Lines 4–10 ensure that the partition's size is greater than a prescribed tolerance and that the partition lies within the contracted bounds.

In Algorithm 3, lines 1–3 execute Algorithm 1 to tighten the bounds using the feasible solution $(\mathbf{x}_{loc}^*, \mathbf{y}_{loc}^*, \mathbf{z}_{loc}^*)$. Interestingly, on some MINLPs, this process shrank the gap between the upper and lower bounds on some variables to 0. Line 5 initializes the tightened bound domains as the active partitions as illustrated in "iteration-0" of Fig. 2. Lines 6–12 iteratively add dynamic partitions around the current solution $\mathbf{x}_{iter}^*$. Iterations 1 and 2 of Fig. 2 clearly illustrate the partitioning scheme employed in this algorithm. The iterations stop (line 6) when (a) the normalized improvement of the lower bound is less than TOL_{imp},
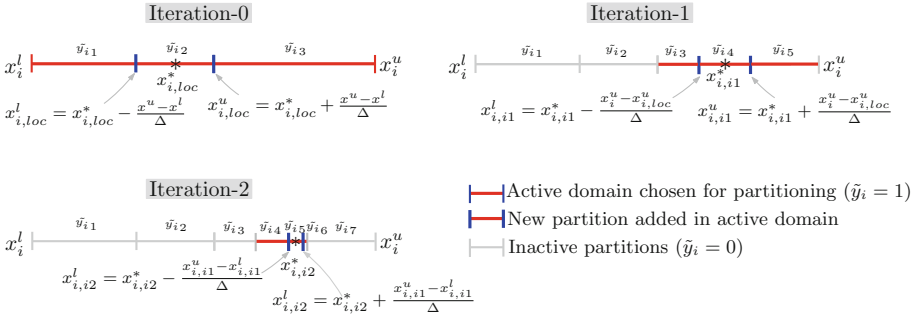
Fig. 2. Dynamic partitioning of variable x_i as described in Algorithm 2

(b) $\mathbf{x}^*_{iter}$ remains in the same partitions and the size of the partitions is $\leq \boldsymbol{\epsilon}$, or (c) the computation hits a time limit. In Fig. 2, the third iteration terminates if the $x^*_{i,i3}$ remains in partition $[x^l_{i,i2}, x^u_{i,i2}]$ and its size is less than ϵ_i. Figure 1(a) is a geometric example of a DTMC iteration applied to a bilinear term. This figure illustrates how the area enclosed by the convex relaxations decreases as partitions are applied.

Algorithm 2. Dynamic partitioning of variable domains

Notation: Let $\boldsymbol{P}^*_{iter}$ represent a vector of active partitions for variable vector $\mathbf{x}$. $\mathbf{x}^l(\boldsymbol{P}^*_{iter})$ and $\mathbf{x}^u(\boldsymbol{P}^*_{iter})$ represent the vectors of lower and upper bounds of the active partitions of $\mathbf{x}$ respectively.

1: Input: $\mathbf{x}^l, \mathbf{x}^u, \mathbf{x}^*_{iter}, \boldsymbol{P}^*_{iter}, \boldsymbol{\epsilon} > 0, \boldsymbol{P}^*_{new} = \emptyset, \Delta > 0$
2: $\mathbf{lb} \leftarrow \mathbf{x}^l(\boldsymbol{P}^*_{iter}), \mathbf{ub} \leftarrow \mathbf{x}^u(\boldsymbol{P}^*_{iter})$
3: Evaluate the size of new partition

$$\mathbf{l}_{iter} = \frac{\mathbf{ub} - \mathbf{lb}}{\Delta}$$

4: **if** $\mathbf{l}_{iter} > \boldsymbol{\epsilon}$ **then**
5:     $\mathbf{v}^l \leftarrow \max(\mathbf{x}^l, (\mathbf{x}^{iter} - \mathbf{l}_{iter})), \mathbf{v}^u \leftarrow \min(\mathbf{x}^u, (\mathbf{x}^{iter} + \mathbf{l}_{iter}))$
6:     $\boldsymbol{P}^*_{new} \leftarrow \{(v^l_i, v^u_i), \forall i = 1, \ldots, n\}$
7:     return $\boldsymbol{P}^*_{new}$
8: **else**
9:     return $\emptyset$
10: **end if**

CP-DTMC Generalization. It is important to note that this approach can be applied to other types of relaxations. For example, consider monomials whose powers contain positive integer exponents (≥ 2). Without loss of generality[2], we assume the monomial takes the form x_i^2. We once again partition the domain of x_i into $N_i \in \mathbb{Z}^+$ disjoint regions. Let $\tilde{\mathbf{y}}_i \in \{0, 1\}^{N_i}$ be the binary variables added to the formulation. Formally, this piecewise convex relaxation, denoted by $\tilde{x}_i \in \langle x_i \rangle^{DTMC-q}$, takes the form:

---

[2] In the case of higher order monomials, i.e., x_i^5, we apply a reduction of the form $x_i^2 x_i^2 x_i \Rightarrow \tilde{x}_i^2 x_i \Rightarrow \tilde{\tilde{x}}_i x_i$.

Algorithm 3. An algorithm for global optimization of MINLPs (CP-DTMC)

1: Input: MINLP, $TOL_{imp} > 0$
2: Obtain local solution $(\mathbf{x}^*_{loc}, \mathbf{y}^*_{loc}, \mathbf{z}^*_{loc})$ for the given MINLP
3: Execute Algorithm 1 $(\mathbf{x}^*_{loc}, \mathbf{y}^*_{loc}, \mathbf{z}^*_{loc})$ to calculate bounds $(\mathbf{x}^l, \mathbf{x}^u)$ on variables $\mathbf{x} \in \mathbb{R}^n$ appearing in multi-linear terms.
4: $\mathbf{x}^*_{iter} \leftarrow \mathbf{x}^*_{loc}, \mathbf{y}^*_{iter} \leftarrow \mathbf{y}^*_{loc}$
5: $\boldsymbol{P}^*_{iter} \leftarrow \{(x^l_i, x^u_i), \forall i = 1, \dots, n\}$ (Initialize the active partitions with the entire domains of variables)

6: **while** Stopping criterion not satisfied **do**
7:     For the current $\mathbf{x}^*_{iter}$ and $\boldsymbol{P}^*_{iter}$, obtain $\boldsymbol{P}^*_{new}$ from Algorithm 2.
8:     $\boldsymbol{P}^*_{iter} \leftarrow (\boldsymbol{P}^*_{iter} \cup \boldsymbol{P}^*_{new})$ (updated partitions for DTMC in line 9)
9:     Solve

$$\boldsymbol{P}_{iter} : \quad \underset{\mathbf{x}, \mathbf{y}}{\text{minimize}} \quad f(\mathbf{x}, \mathbf{y}, \mathbf{z})$$
$$\text{subject to} \quad g(\mathbf{x}, \mathbf{y}, \mathbf{z}) \leq 0,$$
$$h(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0,$$
$$z_K = \langle x_i x_j \dots x_k \rangle^{DTMC}, \quad \forall K \in ML$$
$$\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u,$$
$$\mathbf{y} \in \{0, 1\}^m$$

10:     Let $(\mathbf{x}^*_{iter}, \mathbf{y}^*_{iter})$ be the solution to $\boldsymbol{P}_{iter}$
11:     Update the vector of active partition sets $\boldsymbol{P}^*_{iter}$ such that the binary variable $\hat{y}^*_i$ on x_i is equal to 1.0.
12: **end while**
13: Output: Global optimum solution $(\mathbf{x}^*_{opt}, \mathbf{y}^*_{opt})$ or a lower bound (if solver times out) to the MINLP.

$$\tilde{x}_i \geq x_i^2, \tag{3a}$$

$$\tilde{x}_i \leq \left((\mathbf{x}^l_i \cdot \tilde{\mathbf{y}}_i) + (\mathbf{x}^u_i \cdot \tilde{\mathbf{y}}_i) \right) x_i - (\mathbf{x}^l_i \cdot \tilde{\mathbf{y}}_i)(\mathbf{x}^u_i \cdot \tilde{\mathbf{y}}_i) \tag{3b}$$

$$\sum_{k=1}^{N_i} \tilde{y}_{ik} = 1 \tag{3c}$$

$$\tilde{\mathbf{y}}_i \in \{0, 1\}^{N_i} \tag{3d}$$

Note that $(\mathbf{x}^l_i \cdot \tilde{\mathbf{y}}_i)(\mathbf{x}^u_i \cdot \tilde{\mathbf{y}}_i)$ can be rewritten as $\mathbf{x}^l_i(\tilde{\mathbf{y}}_i \tilde{\mathbf{y}}_i^T)\mathbf{x}^u_i$, where $\tilde{\mathbf{Y}} = (\tilde{\mathbf{y}}_i \tilde{\mathbf{y}}_i^T)$ is an $N_i \times N_i$ symmetric matrix with binary product entries (squared binaries on diagonal). Hence it is sufficient to linearize the entries of the upper triangular matrix with exact representations as discussed in Sect. 2.1. This relaxation is then directly introduced into Algorithm 3. The only modification is to supplement the convex envelops in $\boldsymbol{P}_{iter}$ with these monomial terms.

Lemma 3.1. *Given a finite number of partitions on x_i, the piecewise convex relaxation of $\langle x_i \rangle^{DTMC-q}$ is strictly tighter than $\langle x_i, x_i \rangle^{DTMC}$.*

Proof. For a given, finite number of partitions, N_i, on variable x_i, $\langle x_i, x_i \rangle^{DTMC}$ reduces to the following three-inequalities representing the piecewise convex relaxations:

$$\tilde{x}_i \geq 2(\mathbf{x}_i^l \cdot \tilde{\mathbf{y}}_i)x_i - (\mathbf{x}_i^l \cdot \tilde{\mathbf{y}}_i)^2 \tag{4a}$$

$$\tilde{x}_i \geq 2(\mathbf{x}_i^u \cdot \tilde{\mathbf{y}}_i)x_i - (\mathbf{x}_i^u \cdot \tilde{\mathbf{y}}_i)^2 \tag{4b}$$

$$\tilde{x}_i \leq \left((\mathbf{x}_i^l \cdot \tilde{\mathbf{y}}_i) + (\mathbf{x}_i^u \cdot \tilde{\mathbf{y}}_i)\right) x_i - (\mathbf{x}_i^l \cdot \tilde{\mathbf{y}}_i)(\mathbf{x}_i^u \cdot \tilde{\mathbf{y}}_i) \tag{4c}$$

$$\sum_{k=1}^{N_i} \tilde{y}_{ik} = 1, \ \tilde{\mathbf{y}}_i \in \{0,1\}^{N_i}$$

Clearly, inequalities (4a) and (4b) are under estimators of x_i^2 at grid points $x_i^l, i = 1, \ldots, N_i$ and $x_{N_i}^u$ respectively. The over estimator in (4c) is same as the over estimator defining $\langle x_i \rangle^{DTMC-q}$. Further, the second-order conic under estimator of $\langle x_i \rangle^{DTMC-q}$ can be equivalently represented with infinitely many linear inequalities. However, as discussed above, the under estimators in $\langle x_i, x_i \rangle^{DTMC}$ are finite $(N_i + 1)$, thus relaxing the second-order-cone. Therefore, $\langle x_i \rangle^{DTMC-q} \subset \langle x_i, x_i \rangle^{DTMC}$. □

Because of Lemma 3.1, we use this relaxation rather than McCormick on monomial terms. However, using this relaxation forced us to introduce a technical subtlety into the algorithm implementation. While constraint $\tilde{x}_i \geq x_i^2$ in (3) is a convex, second order cone (SOC), several moderately sized problems were difficult to solve, even with modern, state-of-the-art solvers (CPLEX). Either the solver convergence was very slow or they terminated with a numerical error. To circumvent this issue, we implemented a cutting-plane approach for these constraints. This approach relaxes the SOC constraint with a finite number of valid cutting planes (first order derivatives), produces an outer envelop, and produces a lower bound on the optimal solution. This lower bound is tightened for every violated SOC constraint by adding the corresponding valid cutting plane until a solution obtained is feasible, and hence optimal, for the original SOC set. Figure 1(b) illustrates the outer-approximation procedure. Red colored lines are the under estimators of x_1^2 and the valid cutting planes added to the formulation. In Algorithm 3, this approach is used for the solve routine of line 9. We expect the need for this technical detail to diminish as conic solvers improve.

TCP-DTMC - A hybrid approach. The main idea behind the TCP-DTMC approach is to combine the sequential bound tightening procedure in Algorithm 1 with a three-partition piecewise McCormick relaxation on every variable in multi-linear terms. Since we know $\mathbf{x}_{loc}^*$ from a local solver, we discretize the domain with atmost three partitions and satisfy the rules of partitioning as described in Algorithm 2. Therefore, in line 4(d) of Algorithm 1, the McCormick relaxations are replaced by

$$z_K = \langle x_i x_j \ldots x_k \rangle^{DTMC}, \ \ \forall K \in ML.$$

with an additional constraint,

$$\sum_{k=1}^{3} \tilde{y}_{ik} = 1 \ \forall i = 1, \ldots, |\mathbf{x}|.$$

The primary intuition behind this bound tightening procedure is to obtain tighter bounds around the local solution and possibly converge the bounds to near-optimum solutions in the initial step.

4   Computational Results

All computations were performed using the high performance computing resources at Los Alamos National Laboratory (using nodes for parallel computation) with Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz processors and 62GB of memory. All MILPs were solved using CPLEX 12.6.2 with default options and presolver switched on. All the outer-approximation cutting planes for quadratic terms were implemented as a CPLEX lazy cut callback. BARON 15.2.0 (default options) was the global solver used to benchmark the performances of CP-DTMC and TCP-DTMC. Ipopt 3.12.4 and Bonmin 1.8.4 were used as the local NLP and MINLP solvers, respectively. These solvers were used to produce the initial feasible solution for Algorithm 1. Table 1 summarizes the values of all the parameters used in CP-DTMC. The notation "TO" is used to indicate when the algorithm timed out (time limit=3600 sec) and "GOpt" is used to indicate global optimum, i.e. the lower bound is within 0.0001 % of the known optimal solution. In Table 5, BestΔ and Best$N$ correspond to the best solution found within the CPU limit for DTMC's Δ and UTMC's number of partitions, respectively. Also, in Table 5, we define the following:

$$\%\text{Gap} = \frac{f(\mathbf{x}^*_{opt}, \mathbf{y}^*_{opt}, \mathbf{z}^*_{opt}) - f(\mathbf{x}^*_{iter}, \mathbf{y}^*_{iter}, \mathbf{z}^*_{iter})}{f(\mathbf{x}^*_{iter}, \mathbf{y}^*_{iter}, \mathbf{z}^*_{iter})} \times 100, \quad \%\text{BC} = \frac{\|\mathbf{x}^U - \mathbf{x}^L\|_2 - \|\mathbf{x}^u - \mathbf{x}^l\|_2}{\|\mathbf{x}^u - \mathbf{x}^l\|_2} \times 100$$

In our numerical experiments we considered *three* NLPs and *thirteen* MINLPs that ranged from small, contrived examples to large-scale MINLP benchmark problems selected from MINLPLib 2 [3]. We chose problems whose nonlinearity is expressed with multi-linear terms. The MINLPs chosen for analysis purposes are not exhaustive and we will expand the test-bed in our future work. Table 2 summarizes the statistics of the test-bed including global optimum, number of constraints, binary variables, continuous variables and multi-linear terms. Note that "nlp2" contains two, fourth degree monomial terms and "eniplac" contains bilinear, quadratic and cubic monomials. In the case of the "blend" instances, we partition only a single variable per bilinear term as these were large scale MINLPs[3].

4.1   NLPs

We first consider a small set of simple NLPs, as described in Fig. 3(a) and [6,15,24]. We compare the performance of our algorithms with BARON. These

---

[3] In the "blend" instances, there were few binary variables that appeared in most of the bilinear terms. These are the variables chosen for partitioning.

Table 1. Parameters used in CP-DTMC

|	
N (number of partitions in UTMC)	10, 20, 40
Δ (scaling parameter in DTMC/TCP)	2, 4, 8, 10, 16, 32
Wall time execution limit	3600.0 sec
ϵ (minimum partition length tolerance)	0.001
TOL (bound tightening tolerance)	0.01
TOL_{imp} (% improvement tolerance in DTMC)	0.001 %

Table 2. Problem description

Instance	GOpt	#Constraints	#BVars	#CVars (#CVars-discretized)	#ML
nlp1	58.384	3	0	2(2)	3
nlp2	0	2	0	2(4)	4
nlp3	7049.248	14	0	8(8)	5
ex1223a	4.580	9	4	3(3)	3
ex1264	8.6	55	68	20(20)	16
ex1265	10.3	74	100	30(30)	25
ex1266	16.3	95	138	42(42)	36
fuel	8566.119	15	3	12(6)	3
meanvarx	14.369	44	14	21(7)	28
util	4.305	167	28	117(7)	5
eniplac	-132117.083	189	24	117(24)	66
blend029	13.359	213	36	66(10)	28
blend531	20.039	736	104	168(28)	146
blend718	7.394	606	87	135(20)	100
blend480	9.227	884	124	188(28)	152
blend146	45.297	624	87	135(20)	104

problems are interesting to discuss in more detail. "nlp1", taken from [6], involves both bilinear and quadratic terms. "nlp2" appears in applications related to electromagnetic inverse scattering problems [13]. In this problem, quadrilinear terms in the objective and large bounds on the variables makes it particularly challenging for existing McCormick-relaxation based algorithms. For computational studies, we solve nlp2 in two dimensions ($n = 2$). As shown in Fig. 3, the objective function has multiple global minima at $\pm(1, \sqrt{2})$ and a local minimum at the origin. When solved with IPOPT we get a local solution, $f_{loc}^* = 5$, at $(0,0)$. "nlp3", taken from a standard test-suite [12], has five bilinear terms and large bounds on all the variables. Since this is a challenging problem for the equally

nlp1		nlp2	
minimize $\underset{x_1,x_2}{}$	$6x_1^2 + 4x_2^2 - 2.5x_1x_2$	minimize $\underset{x_1,\dots,x_n}{}$	$\sum_{i=1}^{n}(x_i^2 - i)^2$
subject to	$x_1x_2 \geq 8,$	subject to	$-500 \leq x_i \leq 500,\ i = 1,\dots,n$
	$1 \leq x_1, x_2 \leq 10$		

nlp3

minimize $\underset{x_1,\dots,x_8}{}$ $x_1 + x_2 + x_3$

subject to $0.0025(x_4 + x_6) - 1 \leq 0,$

$0.0025(-x_4 + x_5 + x_7) - 1 \leq 0,$

$0.01(-x_5 + x_8) - 1 \leq 0,$

$100x_1 - x_1x_6 + 833.33252x_4 - 83333.333 \leq 0,$

$x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 \leq 0,$

$x_3x_5 - x_3x_8 - 2500x_5 + 1250000 \leq 0,$

$100 \leq x_1 \leq 10000,$

$1000 \leq x_2, x_3 \leq 10000,$

$10 \leq x_4, x_5, x_6, x_7, x_8 \leq 1000$

(a) Mathematical formulations
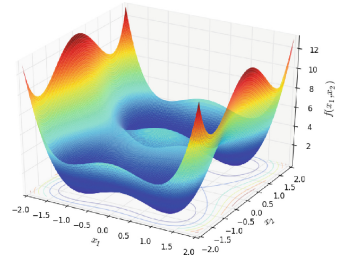
(b) nlp2 with multiple global minima and a local minimum

Fig. 3. NLPs considered in this paper

partitioned, piecewise McCormick relaxations, this problem has been studied in detail in [5,6,24].

Computational Performance. Table 5 summarizes the performance of the algorithms on the NLPs. On nlp1 and nlp2, the algorithms performed consistently better than Baron. For nlp2, we observed that the quadratic convex envelopes, in conjunction with outer-approximation, performed computationally better than solving mixed-integer SOCs.

Table 3. Contracted bounds after applying sequential tightened-CP algorithm on nlp3.

Variable	Original bounds		TCP bounds		#BVars added		
	L	U	l	u	DTMC ($\Delta = 4$)	CP-DTMC ($\Delta = 10$)	TCP-DTMC ($\Delta = 10$)
x_1	100	10000	573.1	585.1	14	14	3+3
x_2	1000	10000	1351.2	1368.5	14	14	3+3
x_3	1000	10000	5102.1	5117.5	17	15	3+3
x_4	10	1000	181.5	182.5	16	15	3+3
x_5	10	1000	295.3	296.0	17	15	3+3
x_6	10	1000	217.5	218.5	16	15	3+3
x_7	10	1000	286.0	286.9	17	15	3+3
x_8	10	1000	395.3	396.0	17	15	3+3
Total					128	118	48

We performed a detailed study of nlp3 as this problem has received considerable interest in the literature. Table 3 show the effectiveness of sequential tightened-CP (TCP) techniques when applied to nlp3. The initial large global bounds are tightened by *a few orders of magnitude* with the addition of three binary variables per continuous variable in the bilinear terms. This shows the value of combining the disjunctive polyhedral approximation around the initial feasible solution ($\mathbf{x}_{loc}^*$) with the bound tightening procedure. In Fig. 4 we also observe that the additional variables do not increase the overall run time too much. More importantly, the reduction in the variable domains is *substantial* using TCP. Finally, the jump in the run time after the first iteration in Fig. 4(b) is due to the reduction in the initial bounds using the CP/TCP algorithm.

Parameter tuning. Table 4 shows the performance of the algorithm on nlp3 for varying values of Δ. It is clear that the solution time and the number of binary variables added in the DTMC algorithm depend on tuning this parameter. However, we note that the % gap for all $\Delta \geq 4$ using TCP-DTMC were close to the optimal solution. For $\Delta = 10$, the global optimum is found in 60 seconds with only 48 binary variables added to the formulation. Overall, for nlp3, it is important to note that the TCP-DTMC algorithm outperforms most of the state-of-the-art piecewise relaxation methods developed in the literature.

(a) Tightened bounds after each iteration

(b) Elapsed time(sec) of bounds tightening
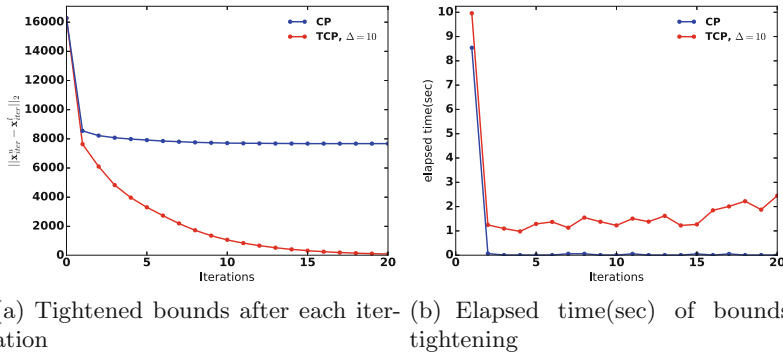
Fig. 4. Performance of sequential CP and sequential tightened-CP on nlp3.

4.2   MINLPs

In this section we compare the algorithms on MINLP benchmark problems described in Table 5.

Performance of DTMC without CP/TCP. From Table 5 it is apparent that dynamically partitioning variable domains to tighten McCormick relaxations is efficient even without bound tightening (CP/TCP). DTMC outperformed the uniform partitioning approach (UTMC) in twelve out of thirteen

Table 4. Performance of proposed algorithms on nlp3 for various Δ values.

Δ	DTMC			CP-DTMC			TCP-DTMC		
	#BVars	T	%Gap	#BVars	T	%Gap	#BVars	T	%Gap
2	137	92.91	141.14	116	1393.31	39.654	160	TO	5.148
4	128	TO	0.013	114	TO	0.032	48	44.44	0.00064
8	116	TO	0.065	117	TO	0.009	48	50.31	0.00014
10	118	TO	0.052	118	TO	0.004	48	59.50	GOpt
16	117	TO	0.092	119	TO	0.009	48	63.04	0.00027
32	118	TO	0.076	120	TO	0.03	48	90.09	0.00029

problems. Problems ex1266, meanvarx and blend718 show the biggest performance gains (UTMC even times out on meanvarx). DTMC also outperforms Baron on ten out of thirteen MINLPs, in particular on eniplac, blend531 and blend718. Blend718 is noteworthy as Baron times out with a 27.5 % optimality gap but DTMC produces global optimum solution within 326.2 sec.

Performance of DTMC with CP/TCP. In Table 5, we observed a reduction in run times of the DTMC algorithm (T_{DTMC}) due to CP/TCP bound tightening (with few exceptions). The reductions are significant on the large-scale blend480 and blend518 problems. Specifically, after TCP, DTMC performs almost twice as fast as Baron on blend480. It is also noteworthy to compare the performance of CP-DTMC and TCP-DTMC with Baron on these problems. We observed that Baron timed out on blend718 and blend146 with 27.5 % and 4.039 % optimality gaps. However, for blend718, CP-DTMC and TCP-DTMC produce global optimum solutions within 488 sec and 208 sec, respectively. On blend146, CP-DTMC and TCP-DTMC timed out with smaller optimality gaps (0.043 % and 0.0570 %) than Baron and UTMC. On blend531, Baron finds the global optimum in 2349 sec, but CP-DTMC and TCP-DTMC find the global optimum in 157 sec and 392 sec - at least fifteen times faster. However, on blend480, while the performance of our algorithms was better than UTMC, it was not better than Baron.

Performance of CP/TCP. Commonly in optimization adding extra binary variables increases problem complexity. However, in Table 5, we observed that the run times for TCP were faster on twelve out of sixteen (including NLPs) instances. Blend480 was an exception, where TCP was almost five times slower than CP. Blend480 is one of the harder MINLPs; it has a large number of binary variables and constraints. From a total domain reduction (BC%) perspective, the advantages of TCP are evident in Table 5. Nlp3, meanvarx and blend029 have the largest reduction. The small BC% values on "blend" problems are due to variable bounds that are tight to begin with.

Performance of convex relaxations on monomials. Table 6 describes the performance of the algorithms when McCormick relaxations ($\langle x, x \rangle^{DTMC}$) are

Table 5. Comparison of all algorithms

Instance	BARON		UTMC			DTMC		
	%Gap	T	Best$N$	%Gap	T$_{UTMC}$	BestΔ	%Gap	T$_{DTMC}$
nlp1	GOpt	4.42	40	0.091	12.74	32	GOpt	1.71
nlp2	GOpt	4.19	20	GOpt	0.07	32	GOpt	0.07
nlp3	GOpt	13.26	40	0.585	TO	4	0.013	TO
ex1223a	GOpt	4.26	20	GOpt	0.02	32	GOpt	0.01
ex1264	GOpt	13.84	10	GOpt	50.62	10	GOpt	1.97
ex1265	GOpt	7.93	10	GOpt	76.35	8	GOpt	0.57
ex1266	GOpt	17.43	10	GOpt	114.15	2	GOpt	0.74
fuel	GOpt	4.38	40	GOpt	1.09	32	GOpt	0.40
meanvarx	GOpt	4.31	40	0.221	TO	8	0.012	90.64
util	GOpt	5.54	40	8.186	6.94	32	0.0098	8.21
eniplac	GOpt	330.46	10	GOpt	2.47	32	GOpt	1.97
blend029	GOpt	15.33	10	GOpt	2.51	32	GOpt	1.95
blend531	GOpt	2348.08	20	0.045	153.43	8	GOpt	140.76
blend718	27.484	TO	20	GOpt	1198.42	16	GOpt	326.17
blend480	GOpt	2044.22	20	0.2	TO	16	0.125	2478.27
blend146	4.039	TO	20	0.58	TO	16	0.035	TO

Instance	CP-DTMC					TCP-DTMC				
	BestΔ	BC(%)	%Gap	T$_{CP}$	T$_{DTMC}$	BestΔ	BC(%)	%Gap	T$_{TCP}$	T$_{DTMC}$
nlp1	16	96.67	GOpt	8.96	1.18	16	98.89	GOpt	2.73	1.10
nlp2	10	99.99	GOpt	8.73	0.02	32	99.99	GOpt	0.34	0.02
nlp3	10	52.86	0.004	9.06	TO	10	99.84	GOpt	59.00	0.50
ex1223a	10	99.00	GOpt	6.31	0.01	10	99.00	GOpt	0.11	0.01
ex1264	10	39.72	GOpt	10.96	1.48	16	40.56	GOpt	5.33	1.74
ex1265	4	23.74	GOpt	10.56	0.64	32	23.74	GOpt	3.20	0.72
ex1266	2	82.29	GOpt	15.15	0.02	4	82.29	GOpt	4.16	0.34
fuel	4	99.90	GOpt	6.95	0.08	4	99.90	GOpt	0.14	0.08
meanvarx	10	67.28	0.004	6.50	12.93	4	84.09	0.0066	8.26	395.23
util	10	99.99	GOpt	13.29	0.47	10	99.99	GOpt	4.73	0.83
eniplac	4	19.15	GOpt	16.71	49.83	32	19.15	GOpt	11.50	5.56
blend029	32	16.08	GOpt	15.73	1.63	10	36.34	GOpt	4.76	1.48
blend531	32	6.91	GOpt	93.91	63.77	4	9.48	GOpt	310.36	82.09
blend718	16	2.38	GOpt	52.07	435.90	32	2.94	GOpt	28.46	179.40
blend480	16	13.89	0.092	183.45	1962.90	16	18.47	0.097	1014.47	1029.00
blend146	32	0.16	0.043	63.71	TO	8	0.45	0.057	30.64	TO

applied to monomial terms. These results are compared with the tighter convex relaxations ($\langle x \rangle^{DTMC-q}$) of Table 5. The run times of DTMC with tighter convex relaxations are faster on all the instances (best on eniplac). Moreover, the total reduction in bounds on variables during CP/TCP steps are up to 11 % larger using tighter convex relaxations.

Table 6. Performance of algorithms with basic McCormick relaxations on higher-order monomials.

Instances with	DTMC		CP-DTMC				TCP-DTMC			
monomials	%Gap	T_{DTMC}	BC(%)	%Gap	T_{CP}	T_{DTMC}	BC(%)	%Gap	T_{TCP}	T_{DTMC}
nlp1	0.0002	0.86	96.67	0.0002	8.16	0.98	98.89	0.00013	1.64	0.37
nlp2	GOpt	13.50	99.99	GOpt	7.99	2.49	99.99	GOpt	0.31	3.74
ex1223a	0.0002	2.16	99.00	0.0001	5.84	0.31	99.00	0.0001	0.79	0.12
fuel	GOpt	1.48	99.82	GOpt	6.45	0.20	99.90	GOpt	3.49	0.21
meanvarx	0.012	755.86	67.28	0.0097	6.43	382.66	74.08	0.0077	6.63	453.31
eniplac	0.0012	350.94	7.68	GOpt	20.31	2662.94	17.26	GOpt	29.98	68.21

5   Conclusions

In this work, we developed an approach for dynamically partitioning McCormick relaxations of multi-linear terms in MINLPs. This is a class of well-known, hard, non-convex optimization problems, where the lower bounds from these relaxations can be arbitrarily bad. We show that a dynamic partitioning of the domains of variables outperforms uniform partitioning and leads to a significantly smaller number of binary variables. We also show that CP techniques, such as bound contraction, can be applied in conjunction with dynamic partitioning to improve convergence drastically. Our numerical experiments suggest that the initial bounds of many benchmark problems are unnecessarily loose, lead to solver scalability issues, and result in poor relaxations.

Finally, we emphasize that the algorithm presented in this paper is by no means exhaustive and there are a number of interesting directions for future research. First, the concept of dynamic partitioning could be combined with tighter convex over and under estimators for nonlinear functions and further improve the quality of the relaxations. Second, we only applied the bound tightening procedure at the root node. We could further apply it at sub nodes not unlike how [19] applies McCormick tightening. Third, there are CP propagation techniques that could be applied to further tighten variable domains. Finally, we could also improve the overall quality of the McCormick relaxations by using different orderings of variables in multi-linear terms.

References

1. Belotti, P., Cafieri, S., Lee, J., Liberti, L.: On feasibility based bounds tightening (2012). https://hal.archives-ouvertes.fr/file/index/docid/935464/filename/377.pdf
2. Bergamini, M.L., Grossmann, I., Scenna, N., Aguirre, P.: An improved piecewise outer-approximation algorithm for the global optimization of MINLP models involving concave and bilinear terms. Comput. Chem. Eng. **32**(3), 477–493 (2008)

3. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib—a collection of test models for mixed-integer nonlinear programming. INFORMS J. Comput. **15**(1), 114–119 (2003)

4. Cafieri, S., Lee, J., Liberti, L.: On convex relaxations of quadrilinear terms. J. Global Optim. **47**(4), 661–685 (2010)

5. Castro, P.M.: Normalized multiparametric disaggregation: an efficient relaxation for mixed-integer bilinear problems. J. Global Optim., 1–20 (2015)

6. Castro, P.M.: Tightening piecewise McCormick relaxations for bilinear problems. Comput. Chem. Eng. **72**, 300–311 (2015)

7. Castro, P.M., Grossmann, I.E.: Optimality-based bound contraction with multi-parametric disaggregation for the global optimization of mixed-integer bilinear problems. J. Global Optim. **59**(2–3), 277–306 (2014)

8. Coffrin, C., Hijazi, H.L., Van Hentenryck, P.: Strengthening convex relaxations with bound tightening for power network optimization. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 39–57. Springer, Heidelberg (2015)

9. Faria, D.C., Bagajewicz, M.J.: Novel bound contraction procedure for global optimization of bilinear minlp problems with applications to water management problems. Comput. Chem. Eng. **35**(3), 446–455 (2011)

10. Grossmann, I.E., Trespalacios, F.: Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming. AIChE J. **59**(9), 3276–3295 (2013)

11. Hasan, M., Karimi, I.: Piecewise linear relaxation of bilinear programs using bivariate partitioning. AIChE J. **56**(7), 1880–1893 (2010)

12. Hock, W., Schittkowski, K.: Test examples for nonlinear programming codes. J. Optim. Theory Appl. **30**(1), 127–129 (1980)

13. Jamil, M., Yang, X.S.: A literature survey of benchmark functions for global optimisation problems. Int. J. Math. Model. Numer. Optim. **4**(2), 150–194 (2013)

14. Karuppiah, R., Grossmann, I.E.: Global optimization for the synthesis of integrated water systems in chemical processes. Comput. Chem. Eng. **30**(4), 650–673 (2006)

15. Kolodziej, S.P., Grossmann, I.E., Furman, K.C., Sawaya, N.W.: A discretization-based approach for the optimization of the multiperiod blend scheduling problem. Comput. Chem. Eng. **53**, 122–142 (2013)

16. Liberti, L., Lavor, C., Maculan, N.: A branch-and-prune algorithm for the molecular distance geometry problem. Int. Trans. Oper. Res. **15**(1), 1–17 (2008)

17. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: part I—convex underestimating problems. Math. Program. **10**(1), 147–175 (1976)

18. Meyer, C.A., Floudas, C.A.: Global optimization of a combinatorially complex generalized pooling problem. AIChE J. **52**(3), 1027–1037 (2006)

19. Mouret, S., Grossmann, I.E., Pestiaux, P.: Tightening the linear relaxation of a mixed integer nonlinear program using constraint programming. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 208–222. Springer, Heidelberg (2009)

20. Ryoo, H.S., Sahinidis, N.V.: Global optimization of nonconvex NLPs and minlps with applications in process design. Comput. Chem. Eng. **19**(5), 551–566 (1995)

21. Sahinidis, N.V.: Baron: a general purpose global optimization software package. J. Global Optim. **8**(2), 201–205 (1996)

22. Smith, E.M., Pantelides, C.C.: A symbolic reformulation/spatial B&B algorithm for the global optimisation of nonconvex MINLPs. Comput. Chem. Eng. **23**(4), 457–478 (1999)

23. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**(2), 225–249 (2005)
24. Teles, J.P., Castro, P.M., Matos, H.A.: Univariate parameterization for global optimization of mixed-integer polynomial problems. Eur. J. Oper. Res. **229**(3), 613–625 (2013)
25. Wicaksono, D.S., Karimi, I.: Piecewise MILP under-and overestimators for global optimization of bilinear programs. AIChE J. **54**(4), 991–1008 (2008)

Parallel Strategies Selection

Anthony Palmieri[1], Jean-Charles Régin[2(✉)], and Pierre Schaus[3]

[1] Huawei Technologie, Boulogne-Billancourt, France
anthony.palmieri@hotmail.fr
[2] University of Nice Sophia Antipolis, CNRS, I3S, UMR 7271,
06900 Sophia Antipolis, France
jcregin@gmail.com
[3] Univ. Louvain-La-Neuven, Louvain-la-Neuve, Belgium
pierre.schaus@uclouvain.be

Abstract. We consider the problem of selecting the best variable-value strategy for solving a given problem in constraint programming. We show that the recent Embarrassingly Parallel Search method (EPS) can be used for this purpose. EPS proposes to solve a problem by decomposing it in many subproblems and to give them on-demand to workers which run in parallel. Our method uses a sample of these subproblems for comparing strategies in order to select the most promising one to be used for solving the remaining subproblems. Each subproblem of the sample is solved with all the candidate strategies in parallel using a timeout that is twice the time of the best one. The selection of the strategy is then based on the Wilcoxon signed rank test. This test is able to deal with censored data caused by timeouts and makes no assumption on the solving time distribution. The experiments we performed on a set of classical benchmarks for satisfaction and optimization problems show that our method selects most of the time the best strategy. Our method also outperforms the portfolio approach consisting of running some strategies in parallel and is competitive with the multi armed bandit framework.

1 Introduction

Many generic variable-value strategies have been imagined [5,8,17,19,23,27]. Those are especially useful in the absence of specific knowledge on the problem to solve. They either try to apply generic principles like the first fail principle (i.e. try to fail as quickly as possible) [13] or try to detect underlined relations between variables and constraints. In the first case, we have strategies like min-domain which selects the variable having the minimum domain size, max-constrained which prefers variables involved in a lot of constraints, or min-regret which selects the variable which may lead to the largest increase in the cost if it is not selected. The latter case is mainly formed by the impact based strategy [23], weighted degree strategy [5] and the activity based strategy [19]. More recently strategies attempting to prioritize variables according to the past failures or conflicting decision have also been designed [8,17,27].

However, selecting *a priori* the best variable-value strategy is not an easy task. Indeed no strategy dominates the other ones in general and it is difficult to identify the types of problems for which a strategy performs well. Any variable-value strategy can give good results for a problem and really bad results for some others. It is not rare to see ratio of performance for a pair of strategy going up from 1 to 20 (and even more sometimes) according to the problems which are solved.

Unfortunately, there is almost no way to compare the performance of variable-value strategies on a problem without solving it. Since strategies explore the search space in different ways and since their pruning performances are not regular it is difficult to compare their behavior before the end of the resolution.

Selecting the right strategy is a challenging decision impacting drastically the solving time.

Our problem can also be seen as the automatic selection of the most efficient algorithm among a predefined set of algorithms, for solving a given problem [15,16,26]. Usually two types of approaches are considered [7]. Either we try to determine statically, that is *a priori*, which is the best algorithm or we dynamically compute the best algorithm to use for each step of the problem solving. Both cases use a set of instances of the problem from which they learn different criteria that will be used to take a decision.

We propose an original approach which is not based on machine learning but on the statistical estimation of the best algorithm. Our approach does not require to deal with a set of instances and use some sampling technique that are usually more accurate. It exploits the decomposition proposed by the embarrassingly parallel search (EPS) method recently developed [24,25].

EPS proposes to solve a problem by decomposing it into a large number of subproblems consistent with the propagation (i.e., there is no immediate failure triggered by the initial propagation of a subproblem). We propose to use a part of these subproblems for comparing the strategies in order to select the most promising one for solving the whole problem. Instead of comparing the strategies after solving the whole problem, we compare the strategies for each subproblem of the sample. We measure the solving time for each subproblem and each strategy and we eliminate the strategies that are statistically proved to be less efficient by a Wilcoxon signed rank test. At the end, either only one strategy remains or a set of non distinguishable strategies. In this latter case we select the one having the smallest mean.

Since for each subproblem the solving times for the strategies may strongly vary, it is necessary to add a timeout mechanism to control the time spent in the strategy selection and to stop some computations after a given amount of time. From a statistical point of view, this means that we may have censored data. By defining appropriately these timeouts, we show that the results of the Wilcoxon signed rank test remains valid if timeouts were not considered. Solving each subproblem with each strategy in parallel allows us to define relative timeouts: we stop a strategy when it requires more than twice the solving time of the best strategy.

It is important to note that our method does not require to know the distribution of the solving times (we made some experiments showing that the

distributions vary according to the problems or to the strategy, and there are no general guidelines).

Our method can be distinguished from the machine learning approaches in two ways:

– The relation between the data from which we take our decision and the instance to solve is stronger in our case because we consider subproblems of the instance and not some other instances.
– We do not try to learn any criteria and we do no try to estimate solving times. We instead aim at selecting the most promising strategy for the given instance only. Our results are statistically validated.

The paper is organized as follows. First we show the principles of our method on an example. Then, we recall some preliminaries. Next, we detail the different steps of our approach. We present some related work and some experiments on a set of benchmarks, for which we compare our results with classical portfolio and a multi-armed bandit method. At last we conclude.

2   Selection Principles

We present the principles of our method on a didactic example obtained from the all-interval series, a common benchmark.

Our method proceeds by elimination of strategies until there is only one remaining.

We consider 4 strategies (S_1, S_2, S_3, S_4). The initial problem has been decomposed into 300 subproblems from which we randomly select only 10 subproblems for the sake of clarity.

We could consider each subproblem in turn and run all the strategies on it in parallel. The drawback of this approach is that the running times are not regular and that some strategies may perform poorly for some subproblems compared to other strategies. For instance, here are the runtimes (in milliseconds) for each subproblem:

Subproblem	S_1	S_2	S_3	S_4
1	62	408	80	150
2	90	1134	92	154
3	155	1904	158	233
4	231	1451	250	407
5	198	1580	197	422
6	146	803	170	144
7	62	611	54	115
8	63	389	111	86
9	167	560	163	670
10	83	736	120	232
Σ	1257	9576	1395	2613

With this approach the total time for selecting the best strategy is $1257 + 9576 + 1395 + 2613 = 14841$, that is more than 10 times the best runtime. Since there are 300 subproblems to solve and since we selected 10, then we can expect a total solving time around 30 times the runtime of the best strategy for our 10 subproblems that is $1.26 \times 30 = 37.8\,\mathrm{s}$[1]. This means that the time allocated to the strategy selection, named selection time, may require more than 40 % of the solving time. In practice running all the strategies on each subproblem in the sample might take up to 90 % of the solving time that would be taken by the best strategy to solve all the subproblems. Our objective is to keep the overhead induced by the selection strategy minimal. Therefore some timeouts are be introduced with respect to the time of the best strategy on each subproblem. Timeouts may cause censored measures that must be carefully treated by statistical methods.

We propose to deal with censored data and proceed by steps.

1. For each subproblem we compare the strategies, but we introduce a timeout limit for each computation corresponding to 2 times the runtime obtained by the best strategy.
2. We select the strategy having the smallest total time (timeouts are counted as their values). If this strategy was stopped by a timeout for some subproblems we run it again on these subproblems without timeouts. We repeat this step until the strategy having the smallest total time without timeout, which we denote s_b, has been selected.
3. We compare all the strategies against s_b by using the Wilcoxon signed rank test. All strategies significantly slower than s_b are eliminated. If s_b is rejected by the Wilcoxon test against s_x (in theory this can happens even if s_b has a better mean) then s_b is eliminated and replaced by s_x. Note that this latter case never happens in the 10,000 s of tests we made.
4. Eventually, if some strategies cannot be distinguished by the Wilcoxon signed rank test then we select the strategy performing the best on the sample.

Note that in any case we have a strong statistical support of our choice.

With timeouts corresponding to twice the runtime of the best strategy for each subproblem we obtain the following table:

#	Timeout	S_1	S_2	S_3	S_4
1	$2 \times 62 = 124$	62	TO	80	TO
2	$2 \times 90 = 180$	90	TO	92	154
3	$2 \times 155 = 310$	155	TO	158	233
4	$2 \times 231 = 462$	231	TO	250	407
5	$2 \times 197 = 394$	198	TO	197	TO
6	$2 \times 144 = 288$	146	TO	170	144
7	$2 \times 54 = 108$	62	TO	54	TO
8	$2 \times 63 = 126$	63	TO	111	86
9	$2 \times 163 = 326$	167	TO	163	TO
10	$2 \times 83 = 166$	83	TO	120	TO
Σ		1257	2 484	1395	2 142

---

[1] We do not claim that this computation is accurate. We present it only for understanding the intuitive idea.

It is important to remark that the best strategy for the whole problem is not the best one for each subproblem. In practice it happens frequently that the best strategy has some timeouts.

The Wilcoxon signed rank test considers the difference in response within pairs. Then it ranks the absolute values of these differences. The sum W^+ of the ranks for the positive difference is the *Wilcoxon signed rank statistic* and has mean $\mu_{W^+} = \frac{n(n+1)}{4}$. The Wilcoxon signed rank test rejects the hypothesis that there is no systematic differences within pairs when the rank sum W^+ is far from its mean.

Suppose we want to compare the strategies S_1 and S_3. For each subproblem we compute the difference $time(S_1) - time(S_3)$. Then, we rank the absolute values of these differences and we add a sign in front of these ranks corresponding of the signs of the differences. For instance, for the first subproblem we have $time(S_1) - time(S_3) = 62 - 80 = -16$, 16 is the 6th values so its rank is 6. The sign rank is -6 because the difference is negative. Then, we compute W^+, the sum of the positive ranks. The following table shows that we have $W^+ = 1 + 5 + 4 = 10$.

Sub problem	S_1	S_3	$S_1 - S_3$	Signed rank
1		62	80	-18	-6
2		90	92	-2	-2
3		155	158	-3	-3
4		231	250	-19	-7
5		198	197	1	$+1$
6		146	170	-24	-8
7		62	54	8	$+5$
8		63	111	-48	-10
9		167	163	4	$+4$
10		83	120	-37	-9

We consider a one-tailed test ($S_3 = S_1$ or $S_3 > S_1$) with a significance level of 0.05.

The critical value of W for $N = 10$ at $p \leq 0.05$ is 10. Therefore the result is significant and we can conclude that S_1 is better than S_3. So, we can eliminate S_3.

We repeat this process between S_1 and the other strategies. We will prove that we can perform the calculations by using the timeouts values if these values are defined by any value greater than twice the maximum positive difference because in this case the positive ranks will not change for any value greater than this timeout. For instance, when we compare S_1 and S_4 there is only one positive difference equal to 2 (for subproblem 6, we have $146 - 144 = 2$), so for each subproblem j we can set the timeout to any value v such that $v > time(S_1, j)$ and $|time(S_1, j) - v| > 2$, because this will not impact the rank of value 2 and so the value of W^+.

If we apply this process for our example, the comparison against S_1 will eliminate all the other strategies.

In conclusion, S_1 is selected. This leads to a resolution time of about 39.4 s.

3   Background

3.1   Statistics

These definitions are due to Moore et al. [21].

Simple Random Samples. A *simple random sample (SRS)* of size n consists of n individuals from the population chosen in such a way that every set of n individuals has an equal chance to be the sample actually selected. We select an SRS by labeling all the individuals in the population and selecting randomly a sample of the desired size. Notice that an SRS not only gives each individual an equal chance to be chosen (thus avoiding bias in the choice) but gives every possible sample an equal chance to be chosen.

Wilcoxon Signed Rank Test for Matched Pairs. Our data do not follow a Normal distribution and timeouts are introduced leading to right censored data. Common method like t-test can thus not be used and non nonparametric tests have to be considered instead for comparing strategies. We use the Wilcoxon Signed Rank Test. Bootstrap methods and permutation tests based on the idea of applying the method many times would be too time-consuming for our purpose.

Since we aim at comparing the performance of two algorithms we consider a *matched pairs* design, which compares just two observations. The idea is that matched subjects are more similar than unmatched subjects, so comparing responses within a number of pairs is more efficient than comparing the responses of groups of randomly assigned subjects. Matched pairs data are analyzed by taking the difference within the matched pairs to produce a single sample. The one sample statistic is applied on this difference data in order to compare the matched pairs data.

The *Wilcoxon signed rank test* (WSR test) for matched pairs is defined as follows. Draw an SRS of size n from a population for a matched pairs study and take the difference in responses within pairs. Rank the absolute values of these differences. The sum W^+ of the ranks for the positive difference is the *Wilcoxon signed rank statistic*. If the distribution of the responses is not affected by the different treatments within pairs, then W^+ has mean $\mu_{W^+} = \frac{n(n+1)}{4}$ and standard deviation $\sigma_{W^+} = \sqrt{\frac{n(n+1)(2n+1)}{24}}$. Difference of zero are discarded before ranking. Ties among the absolute differences are handled by assigning average ranks.

The WSR test rejects the hypothesis that there is no systematic difference within pairs when the rank sum W^+ is far from its mean.

P-values (i.e., the probability computed assuming that null hypothesis is true, that the test statistic will take a value at least as extreme as that actually observed) for the signed rank test are based on the sampling distribution of W^+ when the null hypothesis is true. P-values can be computed from the exact distribution (from software or tables) or obtained from a Normal approximation with continuity correction.

3.2   Embarrassingly Parallel Search [25]

The idea of the Embarassingly Parallel Search (EPS) is to decompose statically the initial problem into a huge number of subproblems that are consistent with propagation (i.e., running the propagation mechanism on them does not detect any inconsistency). These subproblems are added to a queue which is managed by a master. Then, each idle worker takes a subproblem from the queue and solves it. The process is repeated until all the subproblems have been solved.

The decomposition is made by selecting a set V of k variables and then by searching all instantiations of V that are consistent with propagation. There is no specific variable-value strategy used to find these instantiations. The number of generated subproblems depends on the size of V which is determine by successive computations.

The assignment of the subproblems to workers is dynamic and there is no communication between the workers. EPS is based on the idea that if there is a large number of subproblems to solve then the resolution times of the workers will be balanced even if the resolution times of the subproblems are not. In other words, load balancing is automatically obtained in a statistical sense. Interestingly, some experiments [24] have shown that the number of subproblems does not depend on the initial problem but rather on the number of workers. Moreover, they have shown that a good decomposition has to generate more than 30 subproblems per worker.

4   Method

4.1   Simple Random Sample

We use EPS to decompose the initial problem into a huge set of subproblems. Thus the population is the set of these subproblems. The SRS is built by selecting randomly k subproblems from the set of subproblems. The sample is limited to 1% of the subproblems to avoid spending too much time for the strategy selection. If $k = 30$ subproblems seems to be the minimum number of subproblem to consider, then we need to have at least $3,000$ subproblems.

4.2   Comparison of Strategies

Strategies are compared by using the WSR test on the SRS previously defined. For each subproblem of the SRS we run the strategies in parallel and we stop the

slowest ones when they require twice the time of the best strategy. Then, we select the strategy having the smallest sum of solving times for all the subproblems of the SRS. If this strategy was stopped by a timeout for some subproblems we run it again on these subproblems without timeout. We repeat this step until the strategy, denoted by S_b, having the smallest total time without timeout has been selected. Next, we compare all the strategies against S_b by using the WSR test performed on some modified data. All strategies significantly slower than S_b are eliminated. If at a moment, the strategy S_b is rejected by the Wilcoxon test against another strategy S_x, then timeouts are removed for S_x and we use a t-test for deciding whether S_x should become the best strategy. In this latter case we simply replace S_b by S_x.

In any case, we have a strong statistical support of our selection.

Our hypotheses are

H_0: there is no difference between data of both Strategies.

H_a: scores are systematically higher for the second Strategy.

In order to make sure that the result of the test remains valid when exact solving times are considered instead of timeout values, we proceed as follows. Suppose we compare S_b and S_i. Let us show that if we set for each subproblem j the timeout to a value $to(j) > d_{bi}^{\max} + time(S_b, j)$ where $d_{bi}^{\max}$ the largest positive value of $time(S_b) - time(S_i)$ for all the subproblems of the SRS then the test is valid if exact solving times are considered instead of timeouts.

Property 1. *Let $d_{bi}^{\max}$ the largest positive value of $time(S_b, j) - time(S_i, j)$, and $rank(d_{bi}^{\max})$ be its rank in the WSR test of that value. Then, $rank(d_{bi}^{\max})$ is the greatest value of W^+ and for any value v such that $rank(v) > rank(d_{bi}^{\max})$ we have $time(S_b, j) - time(S_i, j) < 0$ and $v > d_{bi}^{\max}$.*

Proof: By definition of the ranks and since $d_{bi}^{\max}$ is the largest positive value of $time(S_b, j) - time(S_i, j)$ then it has the largest rank in W^+, thus any value having an absolute value greater than $d_{bi}^{\max}$ is negative and has a greater rank ⊙

Property 2. *Suppose that for any subproblem j the timeout for j is set for S_i to a value $to(j) > d_{bi}^{\max} + time(S_b, j)$ and let W^+ be the sum computed with these timeouts. Then, for any value of timeout greater than $to(j)$ the value of W^+ remains unchanged.*

Proof: If the timeout is set to $to(j) > d_{bi}^{\max} + time(S_b, j)$ then for any j reaching the timeout $|time(S_b) - time(S_i)| > d_{bi}^{\max}$. From Property 1 the increase of $to(j)$ will not change the rank of the elements of W^+ so the property holds ⊙.

So, for each subproblem j such that S_i has been stopped by a limit which is less than $d_{bi}^{\max} + time(S_b, j)$, we solve again this subproblem with S_i with the time limit defined by $d_{bi}^{\max} + time(S_b, j) + 1$. Therefore, our deduction are statistically valid.

At the end, it is possible that we cannot deduce that some strategies are statistically different. However, this means that they should lead to equivalent

solving time for the whole problem, so we can select any of them. In this case, we select the strategy performing the best on the sample.

If we compare s strategies with an initial timeout fixed to twice the time of the best strategy and if $tmax(S_b)$ denotes the largest solving time of a subproblem of the sample by the best strategy S_b, then the sum of the solving times for all the strategies for each problem in the sample is bounded by $s \times tmax(S_b)$.

Significance Level of the Results. The significance level of the method is bounded by the product of the confidence intervals of each comparison. This means that for k comparisons, each with a confidence interval of 99 %, the overall result has a confidence interval of 0.99^{k-1}. Fortunately we have only few strategies. For instance for 7 strategies, this leads to a confidence level of $0.99^6 = 94\%$. This is quite acceptable.

Optimization Problems. In optimizations problems, an optimal value of an objective function has to be found, thus bounds on this function are important. For each subproblem, all strategies have the same bound. When a bound is found for subproblem i, it is used for all subproblems considered after i for all strategies

5   Related Work

There has been a significant amount of work on automatically selecting or adapting the search strategy. Some successes have been obtained by running some algorithms in parallel in CP [10] and in SAT [12]. Offline and online machine learning based methods are popular. Offline methods select automatically the strategy among a set of available strategies. They perform a learning phase on a training set of instances. They haven been initially proposed for SAT [28] and then for CSPs [22]. Hamadi [11] proposed two methods: continuous search which aims at finding the best strategy for solving a given problem and autonomous search which aims at finding the best strategy in general. These methods are based on machine learning techniques. On the other hand, online methods have been considered. Epstein et al. [6] proposed Adaptive Constraint Engine (ACE), a method which gathers the decision made by several strategies and proceed to a vote in order to decide which one will be applied for the next decision. Gagliolo and Schmidhuber [7] allocate times to each algorithms by using a multi-armed bandit algorithm whose decisions is based on the previous computations. Arbelaez et al. [1] apply Support Vector Machines to the problem of automatically adapting the search strategy of a CP solver in order to more efficiently solve a given instance. Loth et al. [18] define the best strategy during the search by using a multi-armed bandit approaches combined with Monte Carlo Tree Search. Racing algorithms using a non-parametric test based on ranking to successively discard unpromising configurations, like F-Race [3], have also been proposed. However, F-Race does not deal with censored data: it successively executes the algorithm until its completion on new sampled problems. There is no parallel execution and no time-out that is central to our approach.

For a good introduction to Algorithm selection we encourage the reader to refer to [7,16].

6   Experiments

All the experiments have been run in parallel on a parallel machine. The scaling of the EPS method does not depend on the problem solved, so it is the same for all the variable-value strategies. Therefore, for each strategy we have used the sum of the time spent on each core allocated to this strategy as a measure of the time required by the strategy. The best of these times correspond to the value we want to minimize, thus our experiments are based on these times.

Machines. All the experiments have been made on a Dell machine having four E7-4870 Intel processors, each having 10 cores with 256 GB of memory and running under Scientific Linux.

Solver. We implemented our method on the top of Gecode 4.2 (http://www.gecode.org/).

Considered Strategies. After some experiments we selected 7 candidate strategies. Each strategy is dynamic:

- FF implements the first fail principle by selecting the variable with the minimum domain size [13];
- Act selects the variable with the maximum of activity[2] [19];
- Wdegm selects the variable with the maximum weighted degree[3] [4];
- WdegM same as above excepted that the value is selected differently;
- MRegret selects the variable for which the difference between the largest and second-largest value still in the domain is maximum [9].
- MostC selects the most constrained variable.
- D/Wdeg selects the variable for which the ratio of the size of its domain by its weighted degree is minimum [4,5].

After selecting the variable, all strategies but Wdegm, assign to it the minimum value of its domain. WdegM assigns to it the maximum value of its domain. We did not consider impact based strategy [23] because this strategy is not implemented in Gecode.

---

[2] Roughly the activity is defined by the number of times the variables has been introduced in the propagation queue. The activity is increased at most by one for each decision.

[3] The weighted degree of a variable is defined by a counter associated with it. Each time a constraint fails, the counter of each variable involved in the constraint is increased by one.

Benchmarks Instances. We present the most representative results that we obtained (results for other problems are equivalent).

Problems come from the CSPLib, the minizinc challenge [20] or the Hakank's constraint programming blog [14].

For satisfaction problems we search for all solutions and we consider the following problems: all-I: All intervall series 14; Costa: Costa Array 13; Filo: Filomino 13; Lams 9; Qgrp: Quasi group 7; Msplt: Market split s5–08; Sched: sport scheduling 12; Tank: tank attack puzzle 7; Gol: Golomb 12; Perm: Permutation 12.

For optimization problems, we search for the optimal solution and we prove the optimality. Results are given for the following problems: Crew; Dud: dudney thea; Java: java routing trip 6-3; mario; mario medium 3; Fback: minimum feed back; matching problem Money: money change 27; War: War Peace 8; Sugi: Sugiyama 77;

Sampling. The initial problem is decomposed into 16,635 subproblems from which we randomly select 100 subproblems.

6.1   Main Results

PSS denotes the Parallel Strategies Selection that we propose.

Times are expressed in minutes and correspond to the sum of the times spent by all the cores. Bold times indicate the best strategy for the considered problem.

	FF	Act	Wdegm	WdegM	MRegret	MostC	D/Wdeg	PSS
All-I	26.3	210	55.1	54.4	31.6	26.1	**0.8**	0.9
Costa	**46.2**	365	78.2	153	213	41.7	96.9	49.2
Filo	427	160	**12.0**	78.2	335	654	23.5	12.4
Lams	**58.6**	802	416.3	319.2	49.9	48.7	1301	62.0
Qgrp	36.7	41.0	367	877	4.6	3.3	**2.8**	3.0
Msplt	525	1035	616	620	526	**492**	703	515
Sprt	55.8	265	124	116	73.0	36.6	**14.9**	15.4
Tank	29.6	1091	27 K	47 K	40.6	13 K	**3.8**	4.1
Gol	341	295	543	455	183	334	**168**	176
Perm	234	177	159	201	121	331	**27.3**	28.1

In terms of ratio with respect to the best time (i.e., each time is divided by the best time), we obtain the following table which clearly shows the strong disparities between strategies, and that the performance of PSS is close to the one of the best strategy for each problem. We use the following notation: $\bar{x}$ is the mean and geo $\bar{x}$ is the geometric mean.

	FF	Act	Wdegm	WdegM	MRegret	MostC	D/Wdeg	PSS
All-I	32	254	67	66	38.4	31.7	1	1.06
Costa	1.1	8.8	1.9	3.7	5.1	1.0	2.3	1.06
Filo	35	1.0	13	6.5	27	54	1.96	1.04
Lams	1.2	16.5	8.6	6.6	1.0	1.0	26.8	1.06
Qgrp	13.1	14.7	131	314	1.6	1.2	1.0	1.06
Msplt	1.1	2.1	1.3	1.3	1.1	1.0	1.43	1.04
Sprt	3.7	17.8	8.4	7.8	4.9	2.5	1.0	1.03
Tank	7.9	291	7408	12625	10.8	3576	1.0	1.07
Gol	2.0	1.8	3.2	2.7	1.1	2.0	1.0	1.05
Perm	8.6	6.5	5.8	7.4	4.4	12.1	1.0	1.03
geo $\overline{x}$	5.1	12.1	17.6	19.6	4.4	7.3	1.7	1.05
$\overline{x}$	10.6	61.5	765	1304	9.7	368.3	3.8	1.05

For optimization problems we obtain the following results:

	FF	Act	Wdegm	WdegM	MRegret	MostC	D/Wdeg	PSS
Crew	64	258	85	91	68	**58**	74	61
Dud	**15**	34	40	32	37	17	16.1	16.3
Java	24	35	41	35	**21**	24	108	22.7
Mario	**4.2**	45.9	18.7	9.8	7.4	5.8	5.9	4.6
Fback	**126**	281	379	436	128	131	127	133
Money	**0.6**	0.9	0.9	0.6	0.8	0.6	0.6	0.6
War	185	259	232	250	211	**54**	176	56.4
Sugi	504	154	113	111	381	504	**29.1**	30.1

We can also express them in term of ratios w.r.t. the best time in order to see the relative differences between strategies:

	FF	Act	Wdegm	WdegM	MRegret	MostC	D/Wdeg	PSS
Crew	1.10	4.44	1.46	1.57	1.17	1.00	1.21	1.05
Dud	1.00	2.23	2.60	2.06	2.39	1.12	1.07	1.07
Java	1.12	1.62	1.91	1.64	1.00	1.10	5.14	1.06
Mario	1.0	10.9	4.43	2.33	1.75	1.37	1.40	1.09
Fback	1.00	2.22	3.00	3.45	1.02	1.04	1.01	1.05
Money	1.00	1.57	1.57	1.09	1.35	1.12	1.05	1.05
War	3.45	4.82	4.32	4.65	3.92	1.00	3.26	1.05
Sugi	17.3	5.30	3.90	3.80	13.0	17.3	1.00	1.05
geo $\overline{x}$	1.71	3.35	2.67	2.32	2.01	1.56	1.54	1.06
$\overline{x}$	3.38	4.14	2.90	2.57	3.21	3.13	1.88	1.06

Once again our method gives good results. Note that for all problems the Wilcoxon signed rank test was able to eliminate all strategies against the best one.

Next we give some results for the search of the first solution. The chance plays a role in this case. We consider only problems having few solutions since for problems with many solutions the first one is found during the sampling. Times are in minutes and the last column contains the number of subproblems considered before finding one with a solution (recall that the number of subproblems is 16,635). The subproblems are considered as generated by the decomposition. As can be observed the results are surprisingly good with a very limited footprint with respect to the best strategy.

	PSS	Best strategy	Ratio	#firstSol
Filo	5.56	5.45	1.02	5,283
Msplt	26.8	201	1.33	678
Tank	1.94	1.24	1.57	39
Gol	127	125	1.014	12,400

We report next the mean of 250 experiments obtained by randomly selecting the subproblems. The results obtained by PSS are close to the results of the best strategy:

	PSS	Best strategy	Ratio
Filo	7.09	6.19	1.14
Msplt	172	164	1.05
Tank	2.84	2.66	1.07
Gol	101	95	1.07

6.2 Comparison with a Sequential Approach

We compare the results obtained with PSS against the sequential time of the best strategy (Seq+Best) used for each problem. We give wall clock times in minutes.

	All-I	Costa	Filo	Lams	Qgrp	Msplt	Sprt	Tank	Gol	Perm
PSS	0.024	1.29	0.33	1.63	0.08	13.52	0.40	0.11	4.62	0.74
Seq+Best	1.6	34.6	5.95	38.8	2.1	515	9.8	2.9	135	21
Ratio	28.4	26.8	18.3	23.8	26.5	38.1	24.2	26.7	29.1	28.6

6.3   Comparison with Multi-armed Bandit (MAB) Approach

The Multi-Armed Bandit selector is based on a model defined on a set of k arms, one for each strategy, and a set of rewards $R_i(j)$, where $R_i(j)$ is the reward delivered when an arm i has been chosen at time j. A reward reflects the performance of choosing that arm. The idea is to select for each subproblem a strategy (i.e., an arm) and then to solve the subproblem with this strategy. This will give us a reward inversely related to the solving time. The next selection is based on the sequence of the previous trials. We propose to use the UCB1 policy defined in [2], which selects the arm i that maximizes $p(i) = \overline{R}_i + \sqrt{\frac{2lnm}{m_i}}$, where m is the current number of selection, m_i the number of times i has been selected and $\overline{R}_i$ is the mean of the past rewards of the i arm. This policy prefers the most rewarded strategy but also biases the selection toward less frequently selected strategies (this bias factor increases along the iterations). The main difficulty is the definition of the reward function. We adapt the one of Gagliolo and Schmidhuber [7] which is designed for resource allocation and defined by: $\frac{\ln(t_{\max})-\ln(t_i)}{\ln(t_{\max})-\ln(t_{\min})}$, where $t_{\max}$ and $t_{\min}$ are respectively the maximum and minimum solving time and t_i is the time for solving problem i. Experimentally, we obtained the best results by defining $t_{\max} = 10\mu$ and $t_{\min} = \mu/10$ where μ is the mean of the solving times. With such values we accept some variations and degenerate cases (i.e., very bad solving times) will give only negative rewards. We denote by MAB this method. Here is the comparison with PSS:

	Time		Ratio w.r.t. best	
	PSS	MAB	PSS	MAB
All-I	0.9	2.0	1.06	1.14
Costa	49.2	65.4	1.06	1.41
Filo	12.4	36.8	1.04	3.08
Lams	62.0	102	1.06	1.73
Qgrp	3.0	7.8	1.06	2.77
Mspl	515	548	1.04	1.11
Sprt	15.4	19.8	1.03	1.32
Tank	4.1	12.0	1.07	3.13
Gol	176	243	1.05	1.45
Perm	28.1	31.4	1.03	1.15
geo $\overline{x}$			1.05	1.68
$\overline{x}$			1.05	1.83

The results obtained with PSS are better than with MAB. In addition PSS is more robust. These experiments show that applying the reasoning on subproblems coming from the instance to solve is certainly a good idea.

6.4   Comparison with Portfolio

PSS needs 1172 min for solving all the problems. The Portfolio-x4 method runs in parallel the four best strategies. It requires 3959 min which is not competitive with our method.

We also tried to combine our approach with a portfolio approach. PSS-pfolio2 is the PSS method for which we run in parallel the two best estimated strategies when the difference between them is small. The following results show that it is never interesting to run some strategies in parallel.

	All-I	Costa	Lams	Qgrp	Msplt	Perm
PSS	0.9	49.2	62.0	3.0	515	28.1
PSS-pfolio2	1.6	94.0	114	5.4	917	37.9

6.5   Timeout, Sample Size and Simple Impact

The timeout (TO) may have a huge impact on the selection time as shown by the following table, where "without TO" means that we do not stop any strategy when solving a subproblem.

	with TO	without TO
All-I	0.1	4.9
Costa	3.0	16.1
Filo	0.4	17.5
Lams	3.4	9.6
Qgrp	0.2	2.9
Msplt	22.9	82.8
Sprt	0.5	7.8
Tank	0.3	136
Gol	7.9	38.0
Perm	0.8	5.5

We also performed some experiments with a sample size equals to 30 instead of 100. We do not observe any difference for the selected strategy. The best strategy is selected for all problems.

7   Conclusion

The Embarrassingly Parallel Search method solves a problem by decomposing it into subproblems. In order to select the best variable-value strategy to solve a

problem, we propose to use a part of these subproblems and compare some strategies on them. Then, we select the most promising one by using the Wilcoxon signed rank test. This method, PSS, is simple and does not require a lot of computations. It can easily be used in practice because the time allocated to the strategy selection is under control. Some comparisons with other portfolio approaches show the advantage of our method. We also give a model based on the Multi-armed Bandit algorithm which gives interesting results although inferior and less robust than those of PSS. Finally, it appears that it is better to select only one variable-value strategy than running several in parallel, even if we make some mistakes sometimes.

References

1. Arbelaez, A., Hamadi, Y., Sebag, M.: Online heuristic selection in constraint programming. In: International Symposium on Combinatorial Search (2009)
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**(2–3), 235–256 (2002)
3. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9–13 July 2002, pp. 11–18 (2002)
4. Boussemart, F., Hemery, F., Lecoutre, C.: Revision ordering heuristics for the constraint satisfaction problem. In: 1st International Workshop on Constraint Propagation and Implementation held with CP 2004 (CPAI 2004), pp. 9–43, Toronto, Canada, September 2004
5. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: ECAI, vol. 16, p. 146 (2004)
6. Epstein, S.L., Freuder, E.C., Wallace, R.J., Morozov, A., Samuels, B.: The adaptive constraint engine. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 525–542. Springer, Heidelberg (2002)
7. Gagliolo, M., Schmidhuber, J.: Learning dynamic algorithm portfolios. Ann. Math. Artif. Intell. **47**(3–4), 295–328 (2006)
8. Gay, S., Hartert, R., Lecoutre, C., Schaus, P.: Conflict ordering search for scheduling problems. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 140–148. Springer, Heidelberg (2015)
9. Gecode (2012). http://www.gecode.org/
10. Gomes, C.P., Selman, B.: Algorithm portfolios. Artif. Intell. **126**(1–2), 43–62 (2001)
11. Hamadi, Y., Search, C.: From Algorithms to Systems. Springer (2013)
12. Hamadi, Y., Jabbour, S., Sais, L.: Manysat: a parallel SAT solver. JSAT **6**(4), 245–262 (2009)
13. Haralick, R.M., Elliot, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artif. Intell. **14**, 263–313 (1980)
14. Kjellerstrand, H.: My constraint programming blog (2016)
15. Kotthoff, L.: Algorithm selection for combinatorial search problems: a survey. arXiv preprint. arXiv:1210.7959 (2012)

16. Kotthoff, L.: Algorithm selection literature summary (2016). http://larskotthoff. github.io/assurvey

17. Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Last conflict based reasoning. In: ECAI, pp. 133–137 (2006)

18. Loth, M., Sebag, M., Hamadi, Y., Schoenauer, M.: Bandit-based search for constraint programming. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 464–480. Springer, Heidelberg (2013)

19. Michel, L., Van Hentenryck, P.: Activity-based search for black-box constraint programming solvers. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 228–243. Springer, Heidelberg (2012)

20. MiniZinc (2012). http://www.g12.csse.unimelb.edu.au/minizinc/

21. Moore, D.S., McCabe, G.P., Craig, B.A.: Introduction to the Practice of Statistics: Extended Version. W.H. Freeman, New York (2009)

22. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: Irish Conference on Artificial Intelligence and Cognitive Science, pp. 210–216 (2008)

23. Refalo, P.: Impact-based search strategies for constraint programming. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 557–571. Springer, Heidelberg (2004)

24. Régin, J.-C., Rezgui, M., Malapert, A.: Embarrassingly parallel search. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 596–610. Springer, Heidelberg (2013)

25. Régin, J.-C., Rezgui, M., Malapert, A.: Improvement of the embarrassingly parallel search for data centers. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 622–635. Springer, Heidelberg (2014)

26. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)

27. Vilím, P., Laborie, P., Shaw, P.: Failure-directed search for constraint-based scheduling. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 437–453. Springer, Heidelberg (2015)

28. Lin, X., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-based algorithm selection for sat. J. Artif. Intell. Res. **32**, 565–606 (2008)

Learning Parameters for the Sequence Constraint from Solutions

Émilie Picard-Cantin[1(✉)], Mathieu Bouchard[2], Claude-Guy Quimper[1], and Jason Sweeney[2]

[1] Université Laval, 2325, Rue de L'Université, Québec G1V 0A6, Canada
`emilie.picard-cantin.1@ulaval.ca, claude-guy.quimper@ift.ulaval.ca`
[2] PetalMD, 350 Boulevard Charest Est, Québec G1K 3H5, Canada
`{mbouchard,jsweeney}@petalmd.com`

Abstract. This paper studies the problem of learning parameters for global constraints such as SEQUENCE from a small set of positive examples. The proposed technique computes the probability of observing a given constraint in a random solution. This probability is used to select the more likely constraint in a list of candidates. The learning method can be applied to both soft and hard constraints.

Keywords: Constraint acquisition · Timetabling · Machine learning · CSP · Global constraints · Solution counting · Markov chain · Soft constraints

1 Introduction

Accurate mathematical modeling requires a specific and complex training process and a lot of scheduling experience as there are as many models as there are problems. This is why modeling automation has become a popular field of study.

In this paper, we propose a statistical approach that detects the parameters of multiple global constraints such as AMONG and SEQUENCE, two common constraints used in timetabling. This approach, based on machine learning, analyzes given positive examples (the only inputs from the user) and determines which constraint better explains these examples.

The first contribution of this paper is a technique to compute the probability of observing a specific SEQUENCE constraint. Constraint candidates (satisfied by all examples) are compared using their individual probability. The candidate with the lowest probability of being observed, if it is not part of the model, is chosen and added to the optimization model. The second contribution is an improvement on solution counting for the REGULAR constraint using a simplified automaton and a matrix representation. The last contribution is a machine learning tool that can be applied to both soft and hard global constraints.

Section 2 describes the problem that motivates our research. Section 3 lists major contributions to the constraint acquisition field. Section 4 details our machine learning approach to detect the most likely SEQUENCE constraint for

a set of positive examples. Section 5 summarizes our results on a timetabling problem supplied by PetalMD, a specialist in medical scheduling.

2   Problem Description

Our research is motivated by a medical timetabling problem. A schedule is a table where rows are associated to employees and columns are associated to days. The cell (e, t) contains the task assigned to employee e on day t. The optimization model has two objectives : assigning the maximum number of tasks and minimizing the deviations from employees' workload targets. Each employee can be assigned at most one task at a time. Let x_{et} be the task assigned to employee e at time t. Let R be the total number of employees and T be the set of all task types. At least $\underline{c}_i$ and at most $\bar{c}_i$ employees must work on task i every day. We use a global cardinality constraint [16], $GCC([x_{1t}, \ldots, x_{Rt}], \underline{c}, \bar{c})$ for each day t to ensure these requirements are met.

Let d be the total number of days in the schedule. An employee can be assigned at least l and at most u tasks taken from a set $V \subseteq T$ within a period of k consecutive days. This limit is imposed by $\text{SEQUENCE}(l, u, k, [x_{e1}, \ldots, x_{ed}], V)$ for every employee e, i.e. on each row of the schedule. Both global constraints AMONG and SEQUENCE were introduced by Beldiceanu and Contejean [1]. The constraint $\text{AMONG}(l, u, [x_{ij}, \ldots, x_{i(j+k-1)}], V)$ ensures that $x_{ij}, \ldots, x_{i(j+k-1)}$ are assigned to values in V at least l times and at most u times. The constraint $\text{SEQUENCE}(l, u, k, [x_{i1}, \ldots, x_{id}], V)$ ensures that $\text{AMONG}(l, u, [x_{ij}, \ldots, x_{i(j+k-1)}], V)$ holds for every subset of k consecutive variables in $\{x_{i1}, \ldots, x_{id}\}$.

This paper addresses the problem of learning parameters l, u, k and V of a SEQUENCE constraint from a small set of positive solutions. In particular, we apply our method on manually completed schedules provided by PetalMD. SEQUENCE is one of the most common constraint, yet the parameters are hard to extract from clients. The automated learning of this constraint will save PetalMD time and money. Typically, clients express their constraints informally and model creation is a long interactive process where scheduling experts present new schedules and clients tell them what is wrong with the new schedule until all constraints and parameters are determined.

3   Background

As explained in Sect. 1, automatic modeling is a popular field of study. In the present section, we list important concepts related to our research.

3.1   Constraint Acquisition

Bessiere et al. in [5,6,8,10] propose an algorithm named *ConAcq* learning constraint networks from positive and negative solutions using version space learning. Version space learning defines the search space for the constraint network

as a set of constraint network candidates (hypothesis). Each hypothesis not satisfied by all positive examples is removed. *ConAcq* encodes each example as a set of clauses where the atoms are taken from the constraint vocabulary of the library of constraints. A solution to the corresponding satisfiability problem is therefore an admissible constraint network.

O'Connel et al. [13] propose an interactive version space algorithm, which creates a first version space from examples given by the user. From one of the hypotheses, the system builds an qualifying example. The user accepts or rejects it, and the version space is updated accordingly. The algorithm terminates when the version space contains a single hypothesis.

Bessiere et al. [4] propose an active learning system named *QuAcq*. *QuAcq* adds one constraint at a time in the network by presenting partial queries, which are classified by the user as positive or negative. *QuAcq* choose queries that satisfy the constraints in the current network and violate at least one constraint in the library until no such queries exist.

Beldiceanu and Simonis [3] propose a constraint acquisition tool they refer to as *Model Seeker*. This tool builds a satisfaction model from positive examples using constraints from the global constraint catalog. The *Model Seeker* creates a list of candidates, all global constraints satisfying the examples, by generating sequences and matching them against the global constraints using the *Constraint Seeker* [2]. The candidates are ordered according to their pertinence, which is computed using multiple criteria, such as solution density and constraint popularity. A dominance check is performed to remove redundant constraints.

3.2   Solution Counting

The constraint REGULAR($[X_i, \ldots, X_n], \mathcal{A}$) [14] forces the word $[X_i, \ldots, X_n]$ to belong to the regular language defined by the deterministic finite automaton $\mathcal{A}$. The automaton $\mathcal{A}$ is composed of a finite list of states $\mathcal{Q}$, an alphabet Σ, a transition set $\delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$, an initial state $q_0 \in \mathcal{Q}$, and a set of final states $\mathcal{F} \subset \mathcal{Q}$ which determine the end of all accepted words. A sequence $X_1, \ldots, X_n$ is accepted by $\mathcal{A}$ if and only if there exists a sequence of states $q_0, \ldots, q_n \in \mathcal{Q}$ such that $(q_{i-1}, X_i, q_i) \in \delta$ for all $i \in \{1, \ldots, n\}$ and $q_n \in \mathcal{F}$.

Zanarini and Pesant [19] use dynamic programming to count the solutions that satisfy REGULAR($[X_1, \ldots, X_n], \mathcal{A}$). Let $\bar{\mathcal{A}}$ be the unfolded version of $\mathcal{A}$ where layer L_i contains states attainable with the subsequence $[X_1, \ldots, X_{i-1}]$, see Fig. 1. L_1 contains the initial state and L_{n+1} contains all final states.
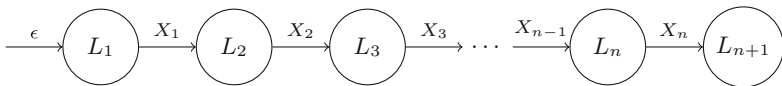
Fig. 1. Unfolded automaton, where L_i contains states attainable with $[X_1, \ldots, X_{i-1}]$.

Let v_{lq} be the state q in layer l and let $\#op(l, q)$ be the number of paths from v_{lq} to a final state in layer L_{n+1}. Then, we have

$$\#op(n + 1, q) = 1 \tag{1}$$

$$\#op(l, q) = \sum_{(v_{l,q}, c, v_{l+1,q'}) \in \delta} \#op(l + 1, q'), \quad \forall q \in \mathcal{Q}, 1 \leq l \leq n. \tag{2}$$

The number of solutions that satisfy REGULAR($[X_1, \ldots, X_n], \mathcal{A}$) is $\#op(1, q_0)$.

Hoeve et al. [11] encode SEQUENCE using REGULAR. Brand et al. [9] improve this encoding by simplifying SEQUENCE($l, u, k, [x_1, \ldots, x_d], V$) to the constraint SEQUENCE($l, u, k, [y_1, \ldots, y_d], \{1\}$) where $\text{dom}(y_i) = \{0, 1\}$ and with the relation $y_i = 1 \iff x_i \in V$. Bessiere et al. [7] show how an automaton can encode the sliding of any constraint over a sequence of variables. Since SEQUENCE($l, u, k, [y_1, \ldots, y_d], \{1\}$) is defined as the sliding of the constraint AMONG on the sequence of variables $[y_1, \ldots, y_d]$, one can get an automaton in the following way. The states are labeled with sequences of zeros and ones of length at most $k - 1$ and are partitioned into two sets. The states in $\mathcal{Q}^s$ are all possible sequences of length $s \leq k - 2$ and are called *transitory states*. They are only visited at the beginning of the sequence. The states in $\mathcal{Q}^{k-1}$ are sequences of length $k - 1$ that contains at least $l - 1$ and at most u occurrences of 1. The initial state is the empty sequence ϵ and the final states are $\mathcal{Q}^{k-1}$.

$$\mathcal{Q}^s = \{0, 1\}^s, \forall s \in \{0, \ldots, k - 2\}, \tag{3}$$

$$\mathcal{Q}^{k-1} = \{w \in \{0, 1\}^{k-1} \mid l - 1 \leq \sum_{i=1}^{k-1} w_i \leq u\}, \tag{4}$$

$$\mathcal{Q} = \bigcup_{i=0}^{k-1} \mathcal{Q}^i. \tag{5}$$

A state $w \in \mathcal{Q}^s$ for $s < k - 1$ leads to the state $wc \in \mathcal{Q}^{s+1}$ upon reading the character $c \in \{0, 1\}$, where wc is the concatenation of the sequence w with the character c. Finally, let a and b be two characters and w a sequence of length $k - 2$. Reading the character b from state $aw \in \mathcal{Q}^{k-1}$ leads to state $wb \in \mathcal{Q}^{k-1}$ only if there are at least l and at most u occurrences of ones in the sequence wb.

$$\delta = \{\langle w, c, wc \rangle \mid w \in \mathcal{Q}^s, wc \in \mathcal{Q}^{s+1}\} \tag{6}$$

$$\cup \{\langle aw, b, wb \rangle \mid aw, wb \in \mathcal{Q}^{k-1}, l \leq a + \sum_{i=1}^{k-2} w_i + b \leq u\}.$$

Figure 2 shows the automaton for SEQUENCE($1, 2, 3, [y_1, \ldots, y_d], \{1\}$). Note that states in $\mathcal{Q}^{k-1}$ are accepting because the transitions only lead to acceptable sequences and the initial state is ϵ, the empty sequence. From the definition of the transitory states, certain states might be isolated in some automatons. Figure 3 illustrates a small example where a state labeled 0 is created but never
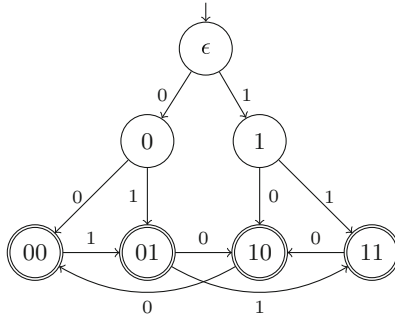
Fig. 2. Automaton corresponding to SEQUENCE$(1, 2, 3, [y_1, \ldots, y_d], \{1\})$
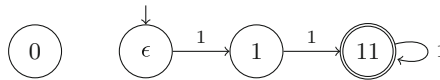
Fig. 3. Automaton corresponding to SEQUENCE$(3, 3, 3, [y_1, \ldots, y_d], \{1\})$

used. We could reduce the automaton by removing those isolated states, but we keep them to simplify notation.

The number of solutions to SEQUENCE can therefore be computed by counting the number of solutions to REGULAR, when used with the automaton that encodes SEQUENCE. As the automaton encodes $v \in V$ with the value 1 and $v \notin V$ with 0, we need a slightly modified version of the solution counting algorithm of Pesant [15] to take into account that there are $|V|$ ways to produce the value 1 and $|T \setminus V|$ ways to produce a 0. We replace Eq. (2) by the following.

$$
\#op(l, q) = \sum_{(v_{l,q}, 0, v_{l+1,q'}) \in \delta} |T \setminus V| \#op(l + 1, q')
$$
$$
+ \sum_{(v_{l,q}, 1, v_{l+1,q'}) \in \delta} |V| \#op(l + 1, q'), \quad \forall q \in \mathcal{Q}, 1 \le l \le n. \quad (7)
$$

For the constraint SEQUENCE$(l, u, k, [y_1, \ldots, y_n])$, since the automaton has $O(2^k)$ states, computing the number of solution is achieved in $O(n2^k)$.

3.3 Markov Chains

Markov chains can be used to compute the number of solutions for REGULAR by encoding the automaton as a transition matrix. A *Markov chain* [17] is a stochastic process defined by a set of time steps $t \in \{0, 1, \ldots, n\}$ and a set of states $i \in \{0, 1, \ldots, m\}$. The variable X_t represent the state of the process at time t. If $X_t = i$, then the process is considered to be in state i at time t. The probability of transitioning from state i to state j is given by a fixed probability P_{ij}. P_{ij} is independent of the states before i. We must have $\sum_{j=1}^{m} P_{ij} = 1$ for all

states i. The transition probabilities are gathered in a matrix P, called the *matrix of one-step transition probabilities* [17]. The n-step transition probabilities P_{ij}^n are the probabilities of being in state j after n transitions if starting in state i. Let α_i be the initial probability of state i. The unconditional probability of ending at state j after n transitions is

$$P[X_n = j] = \sum_{i=0}^{m} \alpha_i P_{ij}^n. \tag{8}$$

4   Constraint Acquisition

In this paper, we propose a statistical learning algorithm, which we divide into three steps. The first step analyzes the given solution, positive example, and makes a list of all constraints satisfied by the solution that call candidates. This is done by verifying each possible constraint against the solution. The second step ranks the selected constraints by computing the individual prior probability of the candidates using Markov chains. The last step chooses the constraint that explains the most the positive example we are given.

4.1   Listing Candidates

The first step of the learning process lists the constraints satisfied by the given solution. We call these satisfied constraints *candidates*, meaning that the real constraint we want to learn is in this subset. To learn the parameters of the constraint $\text{SEQUENCE}(l, u, k, [y_1, \ldots, y_d], V)$, we create all possible sets of parameters and validate them against the solution. The only known parameter is d, since the scope of the constraint is known.

Example 1. Let the number of days be $d = 6$. Let $[y_1, \ldots, y_6] = [1, 1, 0, 0, 1, 1]$ be a solution for which we want to list all $\text{SEQUENCE}(l, u, k, [y_1, \ldots, y_6], \{1\})$ candidates. Since V is known, we refer to each constraint with the tuple (l, u, k).

We have the following candidates. Since each window of length 1 has either 0 or 1 assignments, we have the candidate $(0, 1, 1)$. We have between 0 and 2 assignments for a window of length 2. Therefore, we have the candidate $(0, 2, 2)$. For a window of length 3, we either have 1 assignment or 2, which gives us the candidate $(1, 2, 3)$. Note that we also observe the candidates $(0, 2, 3)$, $(0, 3, 3)$ and $(1, 3, 3)$ which are less restrictive than $(1, 2, 3)$. For windows of length 4, we have the candidates : $\{(l, u, 4) : 0 \le l \le 2 \wedge 2 \le u \le 4\}$. If we continue this process up to $k = d = 6$, we obtain the list

$$\begin{aligned}
C = &\{(0, 1, 1), (0, 2, 2)\} \cup \{(l, u, 3) : 0 \le l \le 1 \wedge 2 \le u \le 3\} \\
&\cup \{(l, u, 4) : 0 \le l \le 2 \wedge 2 \le u \le 4\} \cup \{(l, u, 5) : 0 \le l \le 3 \wedge 3 \le u \le 5\} \\
&\cup \{(l, u, 6) : 0 \le l \le 4 \wedge 4 \le u \le 6\}.
\end{aligned}$$

4.2   Prior Probabilities

To determine which candidate should be learned, we compare them according
to the probability that a random solution validates the candidate constraint.
The best choice is the constraint with the lowest probability because it is highly
improbable that we observe this constraint by chance in the given solution.

Let $\mathrm{dom}(x_j) = \{0, 1, \ldots, m\} = T$ for all $j \in \{1, \ldots, d\}$ and let $p_i = P[x_j = i]$.
Consider $\mathrm{SEQUENCE}(l, u, k, [x_1, \ldots, x_d], V)$ with $V \subset T$ and its simplification
$c = \mathrm{SEQUENCE}(l, u, k, [y_1, \ldots, y_d], \{1\})$. An *acceptable solution according to c* is
a solution that validates c. Let E_c be the random event of observing an acceptable
solution for c. Let X_c be the set of all solutions x satisfying c and let $P[x]$ be
the probability of the specific solution x. Then, we have $P[E_c] = \sum_{x \in X_c} P[x]$.

Inspired from the automaton presented in Sect. 3, which accepts sequences
satisfying a $\mathrm{SEQUENCE}$ constraint, we define a Markov chain $\mathcal{A}'$ that computes
the probability that a random assignment satisfies the sequence constraint. For
the constraint $\mathrm{SEQUENCE}(l, u, k, [y_1, \ldots, y_d], \{1\})$, let the states of the Markov
chain be the set $\mathcal{Q}' = \mathcal{Q}^{k-1} \cup \{\sigma\}$, the sequences of length $k-1$ described in
Sect. 3 and the sink state σ. The states of $\mathcal{Q}^s$ for $s \in \{0, \ldots, k-2\}$ are not
required since a Markov chain does not need an initial state. Let $a, b \in \{0, 1\}$
and $w \in \{0, 1\}^{k-2}$. Then, $aw, wb \in \mathcal{Q}^{k-1}$. There is a transition from aw to wb
with probability $\sum_{i \in V} p_i$ if $b = 1$ and with probability $1 - \sum_{i \in V} p_i$ if $b = 0$.
There is a transition from aw to the sink state σ with probability $\sum_{i \in V} p_i$ if
$a + \sum_{j=1}^{k-2} w_j = u$ and with probability $1 - \sum_{i \in V} p_i$ if $a + \sum_{j=1}^{k-2} w_j = l$. Finally,
there is a transition from σ to σ with probability 1.

Let M be the matrix of one-step transition probabilities for $\mathcal{A}'$. We can
compute, for each state $q \in \mathcal{Q}^{k-1}$, the initial probability g_q of being in q. Let
$[v_{i1}, \ldots, v_{i,k-1}]$ be the sequence of values represented by the state q_i. Then,

$$g_i = \prod_{j=1}^{k-1} p_{v_{ij}} .$$

Note that we can never start with the state σ and therefore $g_\sigma = 0$. Let $g =
[g_1, \ldots, g_r]$ be the initial probabilities for all states $q \in \mathcal{Q}'$. To build a sequence of
length d from the sequences of length $k-1$ (states in $\mathcal{Q}'$), we need the $(d-k-1)$-
step transition probabilities for $\mathcal{A}'$. Then, we have the equation

$$P[E_c] = \sum_{i=1}^{r-1} (gM^{d-k-1})_i . \tag{9}$$

This equation sums the probabilities of each acceptable path in the Markov
chain according to c, a $\mathrm{SEQUENCE}$ constraint. Note that $P[E_c]$ does not include
solutions passing through σ since they represent unacceptable sequences.

Example 2. Suppose we have $c = \mathrm{SEQUENCE}(1, 2, 3, [x_1, \ldots, x_8], \{1, 2\})$ with
$T = \{0, 1, 2\}$ and where $p_0 = 5/6$ and $p_1 = p_2 = 1/12$. The associated Markov
chain is illustrated in Fig. 4. The path from 00 to 00 forms the sequence 000 with

probability $p_0 = 5/6$. This transition violates the constraint. It is redirected to the sink σ. The path from 00 to 01 forms the sequences 001 and 002 and has a probability of $1/6 = p_1 + p_2$. The path from 10 to 01 creates the sequences 101, 102, 201 and $[2, 0, 2]$. This transition has a probability of $p_1 + p_2 = 1/6$. There is a single transition of probability 1 leaving σ. It ensures that paths passing through unfeasible states represented by σ are not counted in $P[E_c]$. The associated matrix of one-step transition probabilities is

$$M = \begin{bmatrix} 0 & 0 & 1/6 & 0 & 5/6 \\ 5/6 & 0 & 1/6 & 0 & 0 \\ 0 & 5/6 & 0 & 1/6 & 0 \\ 0 & 5/6 & 0 & 0 & 1/6 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The initial probabilities are $g = [25/36, 5/36, 5/36, 1/36, 0]$. Therefore, we have

$$P[E_c] = \sum_{i=1}^{4} (gM^4)_i \approx 0.1211 = 12.11\,\%.$$
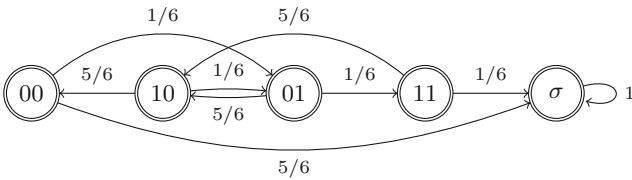
Fig. 4. Markov chain for SEQUENCE$(1, 2, 3, [x_1, \ldots, x_d], \{1, 2\})$ when $p_0 = 5/6$ and $p_1 = p_2 = 1/12$. σ represents all forbidden transitions according to SEQUENCE

As shown in Sect. 3, the solution counting algorithm for the constraint SEQUENCE$(l, u, k, [y_1, \ldots, y_d], \{1\})$ derived from Pesant's algorithm [15] has a computational complexity of $O(d2^k)$. This complexity can be improved. One can compute a power of a matrix using a decrease and conquer approach [12] based on this recurrence.

$$M^p = \begin{cases} I & \text{if } p = 0 \\ M \times M^{p-1} & \text{if } p \text{ is odd} \\ \left(M^{p/2}\right)^2 & \text{otherwise} \end{cases}$$

This algorithm requires $O(\log p)$ matrix multiplications and squaring. Given that multiplying two $n \times n$ matrices requires $O(n^\omega)$ steps ($\omega = 2.373$ when using William's matrix multiplication algorithm [18]), computing the probability that a random assignment satisfies SEQUENCE$(l, u, k, [y_1, \ldots, y_d], \{1\})$ can be achieved in $O(2^{\omega k} \log(d - k))$ steps.

Another method to compute M^{d-k-1} is to use the spectral decomposition of the matrix $M = \mathcal{V}^{-1}\mathcal{D}\mathcal{V}$, where $\mathcal{V}$ is the matrix of eigenvectors for M and $\mathcal{D}$ is the diagonal matrix such that $\mathcal{D}_{ii}$ is an eigenvalue of M. Therefore, we have $M^{d-k-1} = \mathcal{V}^{-1}\mathcal{D}^{d-k-1}\mathcal{V}$. Since computing the eigenvectors and eigenvalues is done in cubic time, this decomposition computes the probability in $O(8^k)$ time. This complexity does not depend on the number of variables. This last method is preferable in situations where the number of variables d for SEQUENCE is largely superior to the number of variables k for each corresponding AMONG.

4.3   Constraints Ordering

Consider a list of candidates $C = \{c_1, \ldots, c_n\}$. We will demonstrate that the best choice is the constraint with the lowest probability since all constraints in C were observed in the solution set.

Theorem 1. *Let $C = \{c_1, \ldots, c_n\}$ be the list of candidates for a solution. Let E_i be the random event of observing an acceptable solution for the constraint $c_i = \text{SEQUENCE}(l_i, u_i, k_i, [y_1, \ldots, y_d], \{1\})$. The constraint c_i explaining the most the current positive example, meaning that $P[E_1 \wedge \cdots \wedge E_n | E_i]$ is maximum, is such that $P[E_i] \leq P[E_j]$ for all $c_j \in C \setminus \{c_i\}$.*

Proof. $P[E_1 \wedge \cdots \wedge E_n | E_i]$ is the probability of observing a solution satisfying all constraints in C considering that c_i is satisfied. Bayes' theorem gives us

$$
\begin{aligned}
P[E_1 \wedge \cdots \wedge E_n | E_i] &= \frac{P[E_i | E_1 \wedge \cdots \wedge E_n] P[E_1 \wedge \cdots \wedge E_n]}{P[E_i]} \\
&= \frac{P[E_1 \wedge \cdots \wedge E_n]}{P[E_i]} .
\end{aligned}
$$

Therefore, we choose c_i over c_j if and only if

$$
P[E_1 \wedge \cdots \wedge E_n | E_i] \geq P[E_1 \wedge \cdots \wedge E_n | E_j]
$$
$$
\Longleftrightarrow \frac{P[E_1 \wedge \cdots \wedge E_n]}{P[E_i]} \geq \frac{P[E_1 \wedge \cdots \wedge E_n]}{P[E_j]} \quad \Longleftrightarrow \quad P[E_j] \geq P[E_i].
$$

4.4   Multiple Examples

If multiple positive examples are available, we can increase the performance of the learning process. The method is applied on each example individually, generating multiple separate candidate lists. Then, considering all examples are restricted with the same SEQUENCE constraint, we keep only the candidates that are present for all examples. The idea is that the real constraint must be satisfied in all examples.

4.5   Constraint Dominance

We say that a constraint c_1 dominates another c_2, noted by $c_1 \succ c_2$, if all solutions to c_1 are solutions to c_2. Following Beldiceanu et al. [2], we reduce computation time by removing dominated constraints from the list of candidates. The concept of dominance is only applied to constraints with the same scope.

   If $c_1 \succ c_2$, then the probability of observing a solution for c_1 is lower (or equal) than the probability of observing a solution for c_2 since c_1 is more restrictive than c_2. Therefore, the classifier will choose c_1 over c_2 and removing dominated constraints does not affect the final choice of the classifier. Moreover, the dominance check is quick and dominance relations can be computed beforehand and stored. Note that if multiple examples are used, the dominance check is performed after collecting all candidate constraints from every example. Indeed, a constraint dominates another one only if it is a candidate for all examples.
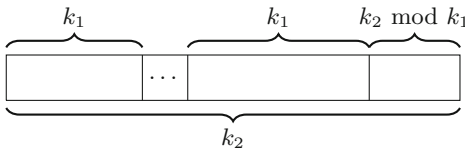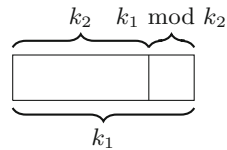
Fig. 5. When $k_1 < k_2$ **Fig. 6.** When $k_1 > k_2$

- If $k_1 = k_2$, then $c_1 \succ c_2$ if and only if $u_1 \leq u_2$ and $l_1 \geq l_2$.
- If $k_1 < k_2$, then a maximum of u_1 assignments of values in V for every window of length k_1 allows a maximum of

$$\bar{a} = \lfloor k_2/k_1 \rfloor \times u_1 + \min(u_1, k_2 \mod k_1)$$

  in a window of length k_2. Similarly, a minimum of l_1 assignments of values in V for every window of length k_1 imposes a minimum of

$$\underline{a} = k_2 - [\lfloor k_2/k_1 \rfloor \times (k_1 - l_1) + \min((k_1 - l_1), k_2 \mod k_1)]$$

  in a window of length k_2. Therefore, $c_1 \succ c_2$ if and only if $\bar{a} \leq u_2$ and $\underline{a} \geq l_2$. See Fig. 5 for a visual representation.
- If $k_1 > k_2$, then $c_1 \succ c_2$ if and only if $(u_1 \leq u_2 \vee k_2 = u_2) \wedge (l_1 \geq l_2 \vee k_2 = l_2)$. See Figs. 5 and 6 for a visual representation.

4.6   Classifier

Algorithm 1 is a summary of the procedure to determine the SEQUENCE constraint from a set of positive examples.

Algorithm 1. How to determine the parameters of a SEQUENCE constraint from a set of positive examples.

Data: Positive examples of d variables with the same configuration.
Result: The parameters of a SEQUENCE constraint.
1 **begin**
2   List all SEQUENCE candidates (l, u, k, V) satisfying all examples;
3   Apply the dominance check to remove dominated constraints;
4   Select, from the remaining candidates, the one with the lower probability of being observed;
5 **end**

4.7 Soft Constraints

The proposed approach can be used to learn soft constraints. The listing of candidates needs to be adapted in order to select constraints which are violated by the given example. Let $\beta \in [0, 1]$ be the accepted percentage of violations. We consider SEQUENCE$(l, u, k, [y_1, \ldots, y_d], \{1\})$ satisfied if at least $(1 - \beta) \times 100\,\%$ of the corresponding AMONG$(l, u, [y_j, \ldots, y_{j+k-1}], \{1\})$ constraints are satisfied by the example.

The Markov chain also needs to be adapted to accept violations. The new set of states is $\mathcal{Q} = \{0, 1\}^{k-1} \cup \{\sigma\}$, the set of all sequences of length $k - 1$ augmented with a sink state σ. The violation degree of a sequence $Y \in \{0, 1\}^k$, according to the constraint AMONG$(l, u, [y_i, \ldots, y_{i+k-1}], \{1\})$, is given by

$$d(Y) = \max\left(\sum_{i=1}^{k} Y_i - u, l - \sum_{i=1}^{k} Y_i, 0\right).$$

Let $w \in \{0, 1\}^{k-2}$, $a, b \in \{0, 1\}$ and $h = \min(d(aw0), d(aw1))$. Let v be the user defined probability of observing $d(awb) > h$, i.e. the probability of reading a character that does not minimize the degree of violation. Let $P[y_i = 1] = \sum_{v \in V} p_v$ and $P[y_i = 0] = 1 - P[y_i = 1]$. If $d(awb) = h$, then we have a transition from state aw to state wb with probability $P[y_i = b]$. If $d(awb) > h$, then we have a transition from aw to wb with probability $vP[y_i = b]$ and a transition from aw to σ with probability $(1-v)P[y_i = b]$. Finally, there is a transition from σ to σ with probability 1.

We assume that the events of accepting the different violated AMONG constraints are independent. Note also that the probability does not depend on the degree of violation of the AMONG constraint, but our model could easily be adapted to do so.

The last modification is the new vector of initial probabilities g. For a state $q \in \mathcal{Q}$, let $r = \sum_{i=1}^{k-1} q_i$. The initial probability for q is

$$g_q = P[y_i = 1]^r \times P[y_i = 0]^{k-1-r} \left(v^{d(0q)} P[y_i = 0] + v^{d(1q)} P[y_i = 1]\right).$$

The initial probability of the sink state σ is $g_\sigma = 1 - \sum_{q \in \mathcal{Q}} g_q$.
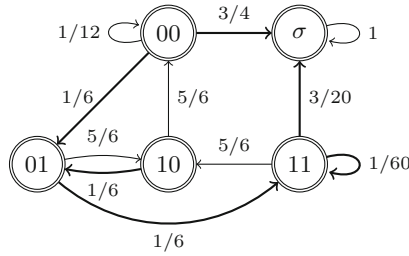
Fig. 7. Graphical representation of the Markov chain corresponding to the soft constraint SEQUENCE$(1, 2, 3, [y_1, \ldots, y_d], \{1\})$ when $v = 1/10$, $p_0 = 5/6$ and $p_1 = 1/6$

Example 3. Suppose $v = 1/10$, $p_0 = 5/6$ and $p_1 = 1/6$. The Markov chain for the soft constraint SEQUENCE$(1, 2, 3, [y_1, \ldots, y_d], \{1\})$ is illustrated in Fig. 7. The initial probabilities for this example are $g_{00} = 25/144$, $g_{01} = g_{10} = 5/36$, $g_{11} = 17/720$ and $g_\sigma = 1 - 342/720 = 378/720$.

5   Experiments

Experiments focus on the timetabling problem presented in Sect. 2. For every instance there are three employees and three medical task types (day, evening, night) repeated each day of an 84 days scheduling period. Tasks are represented by positive integers and 0 is reserved for unassigned days. In our context, the employees want to have similar workloads. As there are 84×3 tasks to assign, the target workload of each employee is 84. An employee e can only work on one task each day. The targets are encoded with soft constraints limiting workloads. We minimize deviations from targets in the objective function. Each task requires one employee, which is encoded using a GCC. There is a constraint SEQUENCE$(0, u, k, [x_{r1}, \ldots, x_{r84}], V)$ where u, k and V are to be learned.

Remember that $x_{et} \in \{0, 1, \ldots, m\}$ is the task assigned to employee e at time t and that $y_{et} \in \{0, 1\}$ determines if $x_{et} \in V$ or not. Let z_{et} be the Boolean variable, which encodes if e is working or not at time t. Therefore, we have $z_{et} = 0$ if $x_{et} = 0$ and $z_{et} = 1$ if $x_{et} > 0$. Let Δ_e be the deviation from the target for the employee e. Let $H \subseteq \{1, 2, 3\}$ be a subset of employees. The general model that produced the instances is as follows.

$$\max \quad \sum_{e=1}^{3}\sum_{t=1}^{84} z_{et} - \sum_{e=1}^{3} 0.1\Delta_e$$

$$84 - \sum_{t=1}^{84} z_{et} \leq \Delta_e, \quad \forall e \in \{1, 2, 3\}$$

$$z_{et} = 1 \iff x_{et} \geq 1, \quad \forall e \in \{1, 2, 3\}, \forall t \in \{1, \ldots, 84\}$$

$$\text{GCC}([x_{1t}, x_{2t}, x_{3t}], [0, 0, 0], [1, 1, 1]]), \quad \forall t \in \{1, \ldots, 84\}$$

$$\text{SEQUENCE}(0, u, k, [x_{e1}, \ldots, x_{e84}], V), \quad \forall e \in H$$
$$x_{et} \in \{0, 1, \ldots, m\}, \quad \forall e \in \{1, 2, 3\}, \forall t \in \{1, \ldots, 84\}$$
$$y_{et}, z_{et} \in \{0, 1\}, \quad \forall e \in \{1, 2, 3\}, \forall t \in \{1, \ldots, 84\}$$
$$\Delta_e \in \mathbb{N}, \quad \forall e \in \{1, 2, 3\}$$

As a benchmark[1], we generate some models and find several optimal solutions for each of them. These solutions are the positive examples that we feed into our algorithm, in order to test whether it returns the constraints that were actually used to generate the solutions. The generated models can be divided into three categories. We create a first set of schedules (A) where the same SEQUENCE constraint is applied to all employees ($H = \{1, 2, 3\}$). Then, we produce five different schedules for all possible combinations of (u, k) with $1 \leq u < k \leq 7$ and $V = \{1, 2, 3\}$. We create a second set of instances (B) where the same SEQUENCE is applied to $e \in \{1, 2\}$. The last employee is not subject to any SEQUENCE constraint. Again, we produce five schedules for all (u, k) with $V = \{1, 2, 3\}$. Finally, we create a last set (C) where SEQUENCE is applied to a subset of tasks and is the same for all employees. For all $V \in \{\{1\}, \{1, 2\}\}$ and for all (u, k), we produce five schedules. Both sets A and B contain 21 instances (105 schedules) and the set C contains 42 instances (210 schedules).

Because of the GCC and the target workloads, all tasks tend to have the same frequency in the schedules. To test our method on instances where values in V do not have the same probability, we create new schedules. For each instance in the sets A, B, and C previously described, we modify the schedules so that each value $i \in V$ has a specific probability p_i to appear in the schedule. Let $I_e = \{t \in \{1, \ldots, 84\} : x_{et} \in V\}$. For $t \in I_e$, we randomly choose a value in V and assign it to x_{et} using on of the following probability distribution: $(P[v_1], P[v_2]) = (0.1, 0.9)$ if $|V| = 2$ and $(P[v_1], P[v_2], P[v_3]) = (0.1, 0.4, 0.5)$ if $|V| = 3$. Then, we apply the same modification process with $I'_e = \{t \in \{1, \ldots, 84\} : x_{et} \in (T \setminus V)\}$. The new schedule still satisfies SEQUENCE since the tasks in V are shuffled between themselves. The GCC might not be satisfied and the solution might not be optimal, but our goal is to test our learning tool on instances with unbalanced distributions of tasks.

The probability of each task is unknown to the learning process. For a given example, we compute the probability of each task with $P[x_{et} = v] = |\{t : x_{et} = v\}|/84$ for each employee e. We approximate the probability $P[v]$ with the average of these probabilities over the five examples. The learning algorithm is applied individually on each employee. We compare the results obtained using this approximation with the results using the solution counting algorithm, which is one of the criteria used by Beldiceanu et al. [3] to rank constraint candidates. We note the statistical learning algorithm *Statistical* and the solution counting version *Counting*. We note *uniform* the instances with uniformly distributed tasks and *non-uniform* the instances with non-uniformly distributed tasks.

The results obtained for only one positive example are illustrated in Table 1. The results are divided by instance set (A, B, or C). # is the total number of

---

[1] The benchmark is available upon request to the authors.
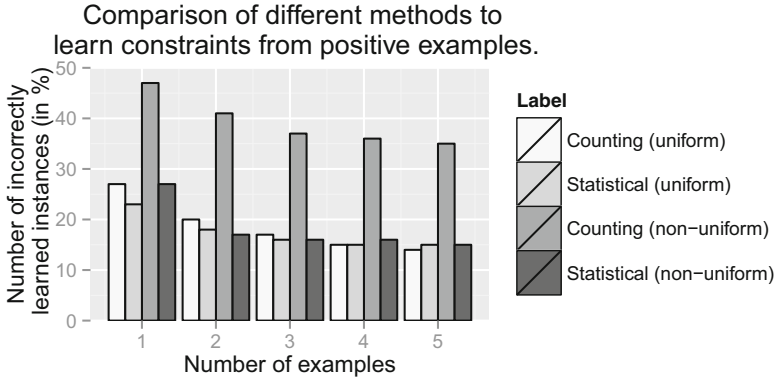
Fig. 8. Number of incorrectly classified instances in percentage for each method and each number of examples

instances in the category (one per employee). Inspired by Beldiceanu et al. [2], we classify our results according to the position of the real constraint in the list of candidates returned.

As shown in Table 1, the first candidate (#1) is the real constraint for all 63 instances of category A while it is the real constraint for 108 out of 126 instances of category C. Table 1 also shows that *Counting* is less efficient on non-uniform instances. For example, 213 uniform instances were correctly learned (ranked #1) while only 164 non-uniform instances were correctly ranked. This is a loss of 49 instances. In comparison, *Statistical* is more stable since it only "lost" 2 correctly ranked instances with the non-uniform task probabilities. This shows that *Statistical* depends on the individual probability of the different values.

Table 1. Results for *Counting* and *Statistical* with a single positive example by instance

| | | Counting | | | | | | | | Statistical | | | | | | | |
| | | Uniform | | | | Non-uniform | | | | Uniform | | | | Non-uniform | | | |
	#	#1	#2	#3	Other	#1	#2	#3	Other	#1	#2	#3	Other	#1	#2	#3	Other
A	63	63	0	0	0	61	1	1	0	63	0	0	0	63	0	0	0
B	63	42	0	0	21	41	1	0	21	42	0	0	21	42	0	0	21
C	126	108	12	1	5	62	7	7	50	108	11	3	4	106	13	2	5
Total	252	213	12	1	26	164	9	8	71	213	11	3	25	211	13	2	26

Figure 8 illustrates the summary results for each method and each number of examples. We can see that, for the instances where tasks are uniformly distributed, *Statistical* is better but *Counting* quickly catches up as the number of examples increases. As illustrated, the lack of uniformity of tasks impacts the performance of both methods. *Statistical* quickly regains the loss with only 4 examples, while *Counting* is still far behind (approximately 20 % apart).

Comparison of different methods to
learn constraints from positive examples.
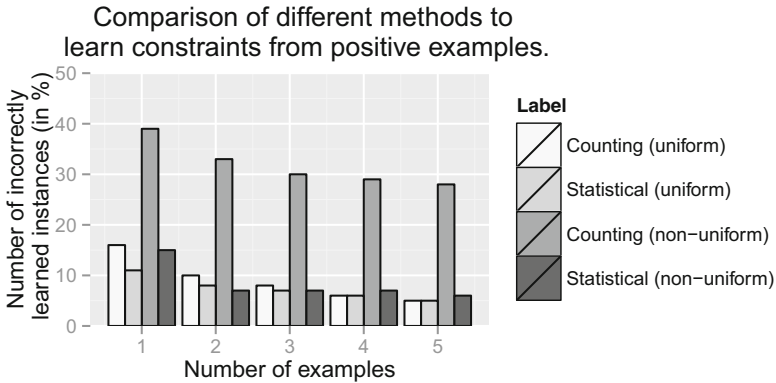
Fig. 9. Number of incorrectly classified instances in percentage after false negatives have been removed

When the real constraint is dominated by one or many candidates, it is removed from the candidate list and the instance is incorrectly classified. We consider this a false negative, as it is impossible to rightly classify this type of instances without further prior information about the problem. Figure 9 illustrates the summary results after the false negatives are removed.

6 Conclusion

In this paper, we proposed a statistical learning algorithm that can be applied to both soft and hard global constraints that can be formulated as an automaton, such as SEQUENCE, AMONG, Knapsack, Stretch, etc. This algorithm uses a new technique to compute the probability of observing a random solution for a given constraint. *Statistical* has proven to be more efficient in the ranking of candidates than the solution counting algorithm, when tested on scheduling instances. For instances where values are uniformly distributed, *Statistical* requires less positive examples to achieve the same results as other methods. This is important in scheduling, where as little as four examples might represent more than a year of data. For instances with non-uniformly distributed values, we showed that *Statistical* is largely better than *Counting*.

References

1. Beldiceanu, N., Contejean, E.: Introducing global constraints in chip. Math. Comput. Model. **20**(12), 97–123 (1994)
2. Beldiceanu, N., Simonis, H.: A constraint seeker: finding and ranking global constraints from examples. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 12–26. Springer, Heidelberg (2011)

3. Beldiceanu, N., Simonis, H.: A model seeker: extracting global constraint models from positive examples. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 141–157. Springer, Heidelberg (2012)

4. Bessiere, C., Coletta, R., Hebrard, E., Katsirelos, G., Lazaar, N., Narodytska, N., Quimper, C.G., Walsh, T.: Constraint acquisition via partial queries. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013), pp. 475–481. AAAI Press (2013)

5. Bessière, C., Coletta, R., Koriche, F., O'Sullivan, B.: A SAT-based version space algorithm for acquiring constraint satisfaction problems. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 23–34. Springer, Heidelberg (2005)

6. Bessiere, C., Coletta, R., Koriche, F., O'Sullivan, B.: Acquiring constraint networks using a sat-based version space algorithm. In: Proceedings of the 21st National Conference on Artificial Intelligence, no. 2, pp. 1565–1568. AAAI Press (2006)

7. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Quimper, C.-G., Walsh, T.: Reformulating global constraints: the SLIDE and REGULAR constraints. In: Miguel, I., Ruml, W. (eds.) SARA 2007. LNCS (LNAI), vol. 4612, pp. 80–92. Springer, Heidelberg (2007)

8. Bessiere, C., Koriche, F., Lazaar, N., O'Sullivan, B.: Constraint acquisition. Artificial Intelligence (2015, In Press)

9. Brand, S., Narodytska, N., Quimper, C.-G., Stuckey, P., Walsh, T.: Encodings of the sequence constraint. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 210–224. Springer, Heidelberg (2007)

10. Coletta, R., Bessiere, C., O'Sullivan, B., Freuder, E.C., O'Connell, S., Quinqueton, J.: Constraint acquisition as semi-automatic modeling. In: Proceedings of the 23rd SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI 2003), pp. 111–124. Springer, London (2004)

11. van Hoeve, W.-J., Pesant, G., Rousseau, L.-M., Sabharwal, A.: Revisiting the sequence constraint. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 620–634. Springer, Heidelberg (2006)

12. Levitin, A.: Introduction to the Design and Analysis of Algorithms. Pearson Education, Newmarket (2011)

13. O'Connell, S., O'Sullivan, B., Freuder, E.C.: A study of query generation strategies for interactive constraint acquisition. In: Bessière, C. (ed.) Applications and Science in Soft Computing. LNCS, vol. 4741, pp. 225–232. Springer, Heidelberg (2004)

14. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 482–495. Springer, Heidelberg (2004)

15. Pesant, G.: Counting solutions of csps: a structural approach. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), pp. 260–265. Morgan Kaufmann Publishers Inc. (2005)

16. Régin, J.C.: Generalized arc consistency for global cardinality constraint. In: Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 1996), vol. 1, pp. 209–215. AAAI Press (1996)

17. Ross, S.M.: Introduction to Probability Models. Elsevier, Oxford (2014)

18. Williams, V.V.: Multiplying matrices faster than coppersmith-winograd. In: Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC 2012), pp. 887–898. ACM (2012)

19. Zanarini, A., Pesant, G.: Solution counting algorithms for constraint-centered search heuristics. Constraints **14**(3), 392–413 (2009)

The PPSZ Algorithm for Constraint Satisfaction Problems on More Than Two Colors

Timon Hertli[1], Isabelle Hurbain[1], Sebastian Millius[1], Robin A. Moser[1],
Dominik Scheder[1,2(✉)], and May Szedlák[1]

[1] ETH Zürich, Zürich, Switzerland
[2] Shanghai Jiaotong University, Shanghai, China
dominik@cs.sjtu.edu.cn

Abstract. The PPSZ algorithm (Paturi et al., FOCS 1998) is the fastest known algorithm for k-SAT. We show how to extend the algorithm and its analysis to (d, k)-Clause Satisfaction Problems where each variable ranges over d different values. Given an input instance with a unique satisfying assignment, the resulting algorithm is the fastest known algorithm for (d, k)-CSP except when (d, k) is $(3, 2)$ or $(4, 2)$. For the general case of multiple satisfying assignments, our algorithm is the fastest known for all $k \geq 4$.

1 Introduction

In its full generality, the Constraint Satisfaction Problem is NP-complete, so most researchers believe that we will never find an efficient algorithm for it. Worse, even getting a substantial edge over trivial exhaustive search is deemed unlikely by most in the community. Far from despairing, people have tried several routes around this. (1) Finding heuristics that work well in practice [3]. (2) Restricting the constraint language (i.e., what types of constraints are allowed). This area evolves around the famous CSP Dichotomy Conjecture by Feder and Vardi [6] and has a strong algebraic flavour (see Krokhin, Bulatov, and Jeavons [12] for a survey). (3) Restricting the *structure* of the instance rather than the constraint language (e.g. Grohe and Marx [9], Szeider [18], Allender, Chen, Lou, Papakonstantinou, and Tang [1]); see Grohe [8] for a survey on both (2) and (3). (4) Coming up with *moderately* exponential algorithms.

The study of moderately exponential algorithms has been especially fruitful in two areas: algorithms for k-satisfiability (short k-SAT) and graph colorability. For example, the algorithm PPSZ solves 3-SAT in $O(1.308^n)$ (Paturi, Pudlák, Saks, and Zane [14], Hertli [10]) instead of the trivial 2^n; Beigel and Eppstein [2] show how to solve 3-colorability in time $O(1.3289^n)$ instead of the trivial 3^n; Björklund and Husfeldt [4] solve k-colorability in time $O(2^n \text{poly}(n))$ instead of the trivial k^n.

We focus on the general constraint satisfaction problem where every variable takes on a value in $[d] := \{1, \ldots, d\}$ and the only structural restriction is that each constraint may depend on at most k variables. Every constraint can be written as the conjunction of at most d^k *clauses*, i.e., disjunctive constraints like $(x_1 \neq 3 \vee x_2 \neq 4 \vee x_1 \neq 2)$. Since d and k are considered constant values, we can re-write an instance as a conjunction of clauses. We call the resulting formula a (d, k)-*clause satisfaction formula* and the corresponding decision problem the *clause satisfaction problem*. We abbreviate both by (d, k)-ClSP. Note that k-SAT is the same as $(2, k)$-ClSP and d-colorability is a special case of $(d, 2)$-ClSP.

In this paper we generalize the PPSZ k-SAT algorithm [14] and Hertli's analysis [10] to (d, k)-ClSP. While it is rather straightforward to adapt the algorithm to handle values $d \geq 3$, analyzing its running time is much more challenging than in the Boolean $(d = 2)$ case. This is in contrast to Schöning's random walk algorithm [17], where both algorithm and analysis generalize easily to $d \geq 3$.

Which Running Time Can We Expect? We measure the running time of an algorithm in terms of n, the number of variables in the input formula F. Trivial brute-force search runs in time $O^*(d^n)$[1], a baseline against which we measure our algorithms. If $P \neq NP$, we will not find a polynomial time algorithm for (d, k)-CSP (expect for the two trivial cases $d = 1$ and $k = 1$, and for $(d, k) = (2, 2)$, which is 2-SAT). Under the Exponential Time Hypothesis [11], there is some $c > 0$ such that every algorithm for $(d, 2)$-ClSP takes time at least $\Omega(d^{cn})$ (Traxler [19]). In other words, (d, k)-ClSP becomes strictly more complex as d increases, even for $k = 2$. This stands in contrast to d-Colorability, which can be solved in $O^*(2^n)$ time [4], for every d. Thus, under the Exponential Time Hypothesis, $(d, 2)$-ClSP is strictly more complex than d-Colorability.

1.1   Previous Results

For k-SAT, the currently fastest known (randomized) algorithm is the PPSZ algorithm by Paturi, Pudlák, Saks and Zane [14]. For instances with a unique satisfying assignment they give an elegant running time analysis. We call this case *UniqueSAT* (or *UniqueClSP* for $d \geq 3$). For the general case (if the instance has multiple satisfying assignments), the analysis becomes much more difficult, and it took over ten years until Hertli [10] showed how to obtain the UniqueSAT time bound in the general case as well.

There are several moderately exponential algorithms for (d, k)-ClSP. For example, a simple random walk algorithm by Schöning [17] solves (d, k)-ClSP in time $O^*\left(\left(\frac{d(k-1)}{k}\right)^n\right)$. Beigel and Eppstein gave an algorithm for $(d, 2)$-ClSP running in time $\mathcal{O}((0.4518d)^n)$ for $d > 3$. Feder and Motwani [5] give an $(d, 2)$-ClSP algorithm based on the PPZ algorithm [15], the predecessor of the PPSZ algorithm, improving on the algorithm by Beigel and Eppstein for large d. Li, Li,

---

[1] The notation $O^*(f(n))$ means $f(n) \cdot 2^{o(n)}$, since we can safely ignore subexponential factors.

Liu, and Xu [13] generalized this to (d, k)-ClSP, but with a sub-optimal weaker analysis. Scheder [16] showed how to use the full power of PPZ for (d, k)-ClSP.

A generic technique for turning any k-SAT algorithm into a (d, k)-ClSP algorithm is *downsampling*: for each variable x in F, randomly forbid all but 2 colors. The resulting instance F' is a $(2, k)$-ClSP and can be solved by any off-the-shelf k-SAT algorithm A. We call this algorithm "downsampling + A". Note that if F is unsatisfiable then F' is; if F is satisfiable then F' is satisfiable with probability at least $(2/d)^n$.

1.2   Our Contribution

We generalize PPSZ to (d, k)-ClSP and analyze its running time. Our upper bound for UniqueClSP is of the form $O^*\left(d^{S_{d,k}n}\right)$ where $S_{d,k} < 1$ is some constant depending on the number of colors d and the arity k of the constraints. We have a complicated but more or less explicit formula for $S_{d,k}$ (involving a sum and an integral). However, there is an intuitive explanation "what $S_{d,k}$ is":

Consider the following random experiment: Let T be an infinite rooted tree in which every even-level vertex (this includes the root, which has level 0) has $k - 1$ children, and every odd-level vertex has $d - 1$ children (there are no leafs). Take $d - 1$ disjoint copies of T, choose a value $p \in [0, 1]$ uniformly at random and delete each odd-level vertex of the $d - 1$ trees with probability p, independently. Let Y be the number of trees in which this deletion still leaves an infinite path starting at the root. Obviously, Y is a random variable and $0 \leq Y \leq d - 1$. Define $S_{d,k} := \mathbf{E}\left[\log_d(1 + Y)\right]$. The full version of this paper will give details on how to compute $S_{d,k}$ more explicitly.

Theorem 1.1. *There exists a randomized algorithm for Unique-(d, k)-ClSP with running time $\mathcal{O}^*(d^{S_{d,k}n})$.*

A randomized algorithm in this context means one that, given a satisfiable input instance, returns a satisfying with probability at least $1/2$. In the general case (when the input formula may have multiple satisfying assignments), we fail to match this running time for $k = 2, 3$. This failure may well be an artifact of our analysis and not reflect the true success probability of PPSZ. Let $G_{d,k} := \max\left(S_{d,k}, 1 - \frac{1}{2\ln(d)}\right)$.

Theorem 1.2. *There exists a randomized algorithm for (d, k)-ClSP with one-sided error that runs in time $O^*(d^{G_{d,k}n})$.*

It turns out that $G_{d,k} = S_{d,k}$ for $k \geq 4$, so for $k \geq 4$ our analysis yields the same performance bounds for the unique and the general case:

Lemma 1.3. *If $k \geq 4$ then $S_{d,k} \geq 1 - \frac{1}{2\ln(d)}$ and therefore $S_{d,k} = G_{d,k}$.*

The proof of Lemma 1.3 is quite technical and contained in the full version of this paper. In the general case, i.e., if F may have multiple satisfying assignments, we also solve an open problem of Hertli [10]: He made a slight (and natural)

Table 1. Constants c so that the algorithm for $(d, 2)$-ClSP runs in time $c^{n+o(n)}$

d	k	PPSZ Unique	PPSZ General	BE [2]	FM [5]	Downsampling+ 2-SAT
3	2	1.434	1.820	**1.356**	1.5	1.5
4	2	1.849	2.427	**1.808**	2	2
5	2	**2.254**	3.033	**2.259**	2.5	2.5
6	2	**2.652**	3.640	**2.711**	2.994	3
10	2	**4.208**	6.066	**4.518**	4.529	5
15	2	**6.115**	9.098	6.777	**6.424**	7.5

Table 2. Constants c so that the algorithm for (d, k)-ClSP runs in time $c^{n+o(n)}$. For (*) $(d, 3)$, $d \geq 11$, PPSZ seems to be worse than PPZ. This is of course not true—PPSZ is subsumes PPZ; it is simply a shortfall of our analysis in Sect. 4.

d	k	PPSZ Unique	PPSZ General	PPZ [16]	Downsampling+PPSZ
3	3	**1.901**	**1.901**	2.162	1.961
4	3	**2.479**	**2.479**	2.729	2.615
5	3	**3.049**	**3.049**	3.291	3.268
10	3	**5.844**	**6.066**	6.069	6.536
11	3	**6.397**	6.672 (*)	**6.621**	7.189
15	3	**8.602**	9.098 (*)	**8.821**	9.803
3	4	**2.153**	**2.153**	2.351	2.204
4	4	**2.823**	**2.823**	3.014	2.938
15	4	**10.006**	**10.006**	10.176	11.018
3	5	**2.310**	**2.310**	2.471	2.355
4	5	**3.040**	**3.040**	3.195	3.139
15	5	**10.906**	**10.906**	11.045	11.771

change to PPSZ but conjectured this change to be unnecessary. We show that this is indeed the case. Furthermore, our proof actually gives a bound on the probability that *a specific satisfying assignment* α is returned, whereas [10] only gave a bound that *some* satisfying assignment is returned (Table 1).

Asymptotics. We want to gauge the performance of several (d, k)-ClSP algorithms for large d. For this, we define the *savings* of an algorithm to be the largest c such that it solves (d, k)-ClSP in time $O^* \left(\frac{d^n}{2^{cn}} \right)$. Note that the enumerator in this definition is 2^{cn}, not d^{cn}. This is because we simply do not know any algorithm that solves (d, k)-ClSP in time $O^* \left(d^{(1-\epsilon_k)n} \right)$ for $\epsilon_k > 0$ independent of d (Table 2).

Theorem 1.4. *For $k \geq 2$ and large d, the savings of PPSZ for (d, k)-ClSP converge to $\log_2(e)(1 - S_{2,k})$, and $1 - S_{2,k} = -\int_0^1 \ln(1 - r^{k-1}) dr$.*

Table 3. The savings of several (d, k)-ClSP algorithms. For PPSZ and PPZ the savings hold for $d \to \infty$. The savings of downsampling+PPSZ and of Schöning do not depend on d.

k	PPSZ (and PPZ)	Downsampling+PPSZ	Schöning
general k	$\log_2(e)(1 - S_{2,k})$	$1 - S_{2,k}$	$\log_2\left(\frac{k}{k-1}\right)$
2	1.44	1	1
3	0.885	0.613	0.585
4	0.642	0.445	0.415
5	0.504	0.349	0.322
$k \to \infty$	$\frac{\pi^2 \log_2(e)}{6k} \approx \frac{2.371}{k}$	$\frac{\pi^2}{6k} \approx \frac{1.644}{k}$	$\frac{\log_2(e)}{k} \approx \frac{1.44}{k}$

This means the savings for large d are a factor $\log_2(e) \approx 1.44$ larger than the savings for k-SAT. It should be mentioned that for large d the advantage of PPSZ over PPZ vanishes, i.e., their savings converge, for each fixed k. A detailed proof of Theorem 1.4 is contained in the full version of this paper. We compare the savings of several algorithms in Table 3.

1.3   Notation

We adapt the notational framework as used in [20]. Let V be a finite set of variables, each of which takes values in $[d] := \{1, \ldots, d\}$. A *literal* over $x \in V$ is of the form $(x \neq c)$ for $c \in [d]$. A *clause* over V is a disjunction (OR) of finite set of literals over pairwise distinct variables from V. A formula F over V is a conjunction (AND) of clauses over V. It is sometimes convenient to view F as a *set* of clauses. By $\mathrm{vbl}(F)$ we denote the set of variables appearing in F. A formula F is a (d, k)-ClSP if the variables can take on d values and every clause has at most k literals. We also write (d, k)-ClSP to denote the satisfiability decision problem on (d, k)-ClSP formulas. By Unique (d, k)-ClSP we denote the promise problem of deciding whether a (d, k)-ClSP has exactly one or no satisfying assignment.

An *assignment* on V is a function $\alpha : V \to [d]$. It satisfies the literal $(x \neq c)$ if $\alpha(x) \neq c$; it satisfies a clause if it satisfies at least one literal therein; finally, it satisfies a formula if it satisfies all its clauses. A *partial assignment* α on V is a partial function $V \to [d]$. It is convenient to view α as a certain $(d, 1)$-CSP over V: for example $(x_1 = c_1) \wedge (x_2 = c_2)$ is the partial assignment that sets x_1 to c_1 and x_2 to c_2. Two partial assignments α, β over V are *compatible* if the $(d, 1)$-CSP $\alpha \wedge \beta$ is satisfiable; in other words, if α and β agree wherever they are defined. For a partial assignment α, we denote by $\mathcal{U}_\alpha$ the set of variables in V on which α is not defined. We denote by $\alpha[x = c]$ the (partial) assignment that sets x to c and agrees with α elsewhere.

By $\models$ we denote usual logical implication. That is, for two formulas F, G over a variable set V, the expression $F \models G$ means that every total assignment α that satisfies F also satisfies G.

2   The PPSZ Algorithm

Definition 2.1 (D-implication). *Let F be a satisfiable ClSP formula over V, α_0 a partial assignment, and $D \in \mathbb{N}$. We say that (F, α_0) D-implies the literal $(x \neq c)$ and write $(F, \alpha_0) \vDash_D (x \neq c)$ if there is a subset G of F with $|G| \leq D$ such that $G \wedge \alpha_0$ implies $(x \neq c)$. Here, $|G|$ is the number of clauses in G.*

Whether $(F, \alpha_0) \vDash_D (x \neq c)$ holds or not can be checked in time $O(|F|^D \cdot d^{kD} \cdot \text{poly}(n))$. If D is constant this is polynomial. If D is sufficiently slowly growing this is subexponential (note that d, k are always assumed to be constant).

Definition 2.2 (Eligible values). *Let F be a satisfiable ClSP formula over V, α_0 a partial assignment, and $x \in \mathcal{U}_{\alpha_0}$ (i.e. an unassigned variable). Then*

$$\mathcal{A}(x, \alpha_0) := \{c \in [d] \mid (F, \alpha_0) \nvDash_D (x \neq c)\} \ .$$

That is, $\mathcal{A}(x, \alpha_0)$ is the set of colors not ruled out by D-implication.

Note that $\mathcal{A}(x, \alpha_0)$ also depends on F and D. However, F and D will not change throughout the analysis, so we will assume from now on that they are clear from the context.

Let us describe PPSZ. Given a ClSP F, it starts with the empty assignment $\alpha_0 = \emptyset$ and attempts to incrementally add variables to it, hoping that eventually α_0 becomes a satisfying (total) assignment. To achieve this, PPSZ chooses a uniformly random permutation π of V and iterates through V in the order dictated by π. When considering some $x \in V$ it computes $\mathcal{A}(x, \alpha_0)$. If this is empty then $F \wedge \alpha_0$ is unsatisfiable and PPSZ declares failure. Otherwise, it chooses some eligible color $c \in \mathcal{A}(x, \alpha_0)$ uniformly at random, adds $(x = c)$ to α_0, and continues. Below we give a pseudo-code for PPSZ. Our pseudo-code is recursive rather than iterative because this is more convenient for the analysis of the general (multiple satisfying assignments) case.

Algorithm 1. Top-Level-PPSZ(ClSP formula F)

Choose π u.a.r. from all permutations of $V(F)$.
Let α_0 be the empty assignment.
return PPSZ(F, π, α_0)

Note that $\mathcal{A}(x, \alpha_0)$ is the set of values that are "currently eligible for x". Now suppose α_0 is compatible with some satisfying assignment α, and the next assignment steps of PPSZ are all according to α. We are actually interested how $\mathcal{A}(x, \alpha_1)$ will look where α_1 is the "future" partial assignment just before x is processed. This motivates the following (recursive) definition.

Definition 2.3 (Ultimately Eligible Values). *Let π a permutation of the variables, α be a satisfying assignment, α_0 a partial assignment compatible with α, and x a variable in $\mathcal{U}_{\alpha_0}$. Let y be the first variable of $\mathcal{U}_{\alpha_0}$ according to π.*

Algorithm 2. PPSZ(F, π, α_0)

> **if** α_0 is a total assignment **then**
>> **return** α_0 if it satisfies F, else `failure`
>
> **end if**
> $x \leftarrow$ first variable of $\mathcal{U}_{\alpha_0}$ according to π
> $c \leftarrow_{\text{u.a.r.}} \mathcal{A}(x, \alpha_0)$ (return `failure` if $\mathcal{A}(x, \alpha_0) = \emptyset$).
> **return** PPSZ(F, π, $\alpha_0 \wedge (x = c)$)

- *If $y = x$ set $\mathcal{A}(x, \alpha_0, \alpha, \pi) := \mathcal{A}(x, \alpha_0)$.*
- *Otherwise, set $\mathcal{A}(x, \alpha_0, \alpha, \pi) := \mathcal{A}(x, \alpha_0 \wedge (y = \alpha(y)), \alpha, \pi)$.*

This definition allows us to write down an explicit formula for the success probability of PPSZ. We write

$$p(\alpha_0, \alpha) := \Pr_{\pi}[\text{PPSZ}(F, \pi, \alpha_0) \text{ returns } \alpha] .$$

This is the probability that PPSZ returns one particular assignment α. Observe that PPSZ returns α if and only if it always picks the "correct" value for every $x \in \mathcal{U}_{\alpha_0}$. For a fixed permutation π this happens with probability $\frac{1}{|\mathcal{A}(x, \alpha_0, \alpha, \pi)|}$. Therefore we obtain

$$p(\alpha_0, \alpha) = \mathbf{E}_{\pi}\left[\prod_{x \in \mathcal{U}_{\alpha_0}} \frac{1}{|\mathcal{A}(x, \alpha_0, \alpha, \pi)|} \right] \tag{1}$$

$$\geq d^{-\sum_{x \in \mathcal{U}(\alpha_0)} \mathbf{E}_{\pi}[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|]} . \qquad \text{(by Jensen's Inequality)}$$

A large part of this paper will be devoted to estimating $\mathbf{E}_{\pi}[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|]$. Note that in general there is no non-trivial upper bound: If F is the empty formula over n variables, which always evaluates to 1, then $\mathcal{A}(x, \alpha_0, \alpha, \pi) = d$ for all x and π and $p(\alpha_0, \alpha) = d^{-|\mathcal{U}_{\alpha_0}|}$. In particular, this is d^{-n} if we start with the empty assignment $\alpha_0 = \emptyset$. In the other extreme, if there is only one possible value of x, we can actually give a non-trivial upper bound.

Definition 2.4 (Frozen Variables). *Let α_0 a partial assignment. A variable $x \in \mathcal{U}(\alpha_0)$ is frozen (in F with respect to α_0) if there is a value $c \in [d]$ such that $F \wedge \alpha_0 \models (x = c)$.*

Here we are talking about "full implication" $\models$, not D-implication $\models_D$.

Lemma 2.5. *Let F be a (d, k)-ClSP formula, α a satisfying assignment, α_0 a partial assignment compatible with α, and x a variable in $\mathcal{U}_{\alpha_0}$. If x is frozen in F with respect to α_0 then*

$$\mathbf{E}_{\pi}[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|] \leq S_{d,k} + \epsilon_D ,$$

where ϵ_D is an error parameter that goes to 0 as D goes to infinity.

This lemma immediately implies Theorem 1.1.

Proof (of Theorem 1.1) Suppose F has exactly one satisfying assignment α. Let α_0 be the empty assignment. Note that every x is frozen in F with respect to α_0.

$$p(\alpha_0, \alpha) \geq d^{-\sum_{x \in \mathcal{U}(\alpha_0)} \mathbf{E}_\pi [\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|]}$$
$$\geq d^{-n(S_{d,k} + \epsilon_D)} \ .$$

By making D a slowly growing function in n, we can make sure that PPSZ runs in subexponential time and has success probability $O^* \left(d^{-S_{d,k}n} \right)$. Repeating this procedure $O^* \left(d^{S_{d,k}n} \right)$ times guarantees a success probability of at least $1/2$.

3   Understanding $|\mathcal{A}(x, \alpha_0, \alpha, \pi)|$: Proof of Lemma 2.5

For this whole section, we fix a partial assignment α_0, a satisfying assignment α of F that is compatible with α_0, and a frozen variable x on which α_0 is not defined. Without loss of generality, we let $\alpha = (d, \ldots, d)$. Since x is frozen we have $F \wedge \alpha_0 \models (x = d)$. Similar to [14] we construct *critical clause trees*.

3.1   Construction of Critical Clause Trees

For each color $c \in \{1, \ldots, d-1\}$ we construct a *critical clause tree* T_c. This is a tree with two types of nodes: *clause nodes* on even levels (this includes the root, which is on level 0) and *variable nodes* on odd levels. A clause node u has a clause label clause-label$(u) \in F$ and an assignment label β_u; it will always hold that β_u is compatible with α_0 and violates clause-label(u); a clause node has at most $k-1$ children. A variable node v has a variable label var-label$(v) \in \mathcal{U}_{\alpha_0}$ and exactly $d-1$ children. Furthermore, each edge $e = (v, w)$ from a variable node v to a clause node w has an edge color edge-color$(e) \in [d-1]$. Here is how we construct T_c:

Create a a root vertex and set $\beta_{\text{root}} := \alpha[x = c]$.
`while` there is a leaf u without a clause label:
– Choose a clause C unsatisfied by β_u.
– Set clause-label$(u) := C$.
– `for all` literals $(y \neq d) \in C$:
  • Create a new child v of u. Set var-label$(v) = y$.
  • `for all` $i \in [d-1]$: Create a new child of w of v and set $\beta_w := \beta_u[y = i]$, edge-color$(v, w) = i$.

Proposition 3.1. *(1) The construction of T_c terminates. (2) Suppose u is a clause node in T_c, $C =$ clause-label(u) and $(y \neq i)$ is a literal in C. If $i = d$ then u has a child v with var-label$(v) = y$. If $i < d$ then u has an ancestor v with var-label$(v) = y$. (3) If var-label$(v) =$ var-label(v') then v is not an ancestor of v'. In other words, the set of variable nodes v with var-label$(v) = y$ is an anti-chain in T_c.*
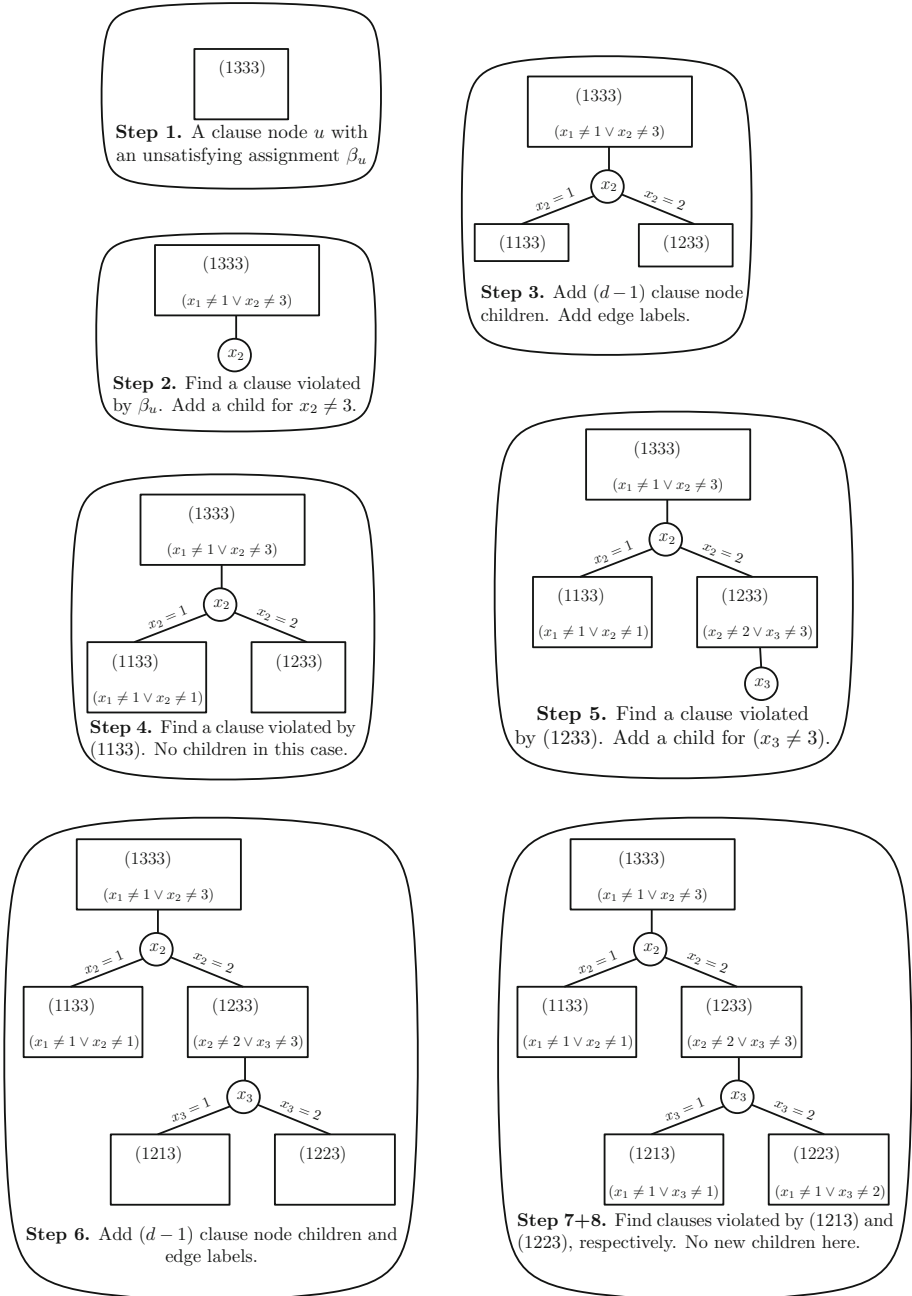
Fig. 1. Construction of a critical clause tree for $d = 3$, $V = \{x_1, x_2, x_3, x_4\}$, $\alpha = (3, 3, 3, 3)$, $\alpha_0 = \emptyset$, variable x_1, and color $c = 1$. The formula F contains, amongst others, the clauses $(x_1 \neq 1 \vee x_2 \neq 3) \wedge (x_1 \neq 1 \vee x_2 \neq 1) \wedge (x_2 \neq 2 \vee x_3 \neq 3) \wedge (x_1 \neq 1 \vee x_3 \neq 1) \wedge (x_1 \neq 1 \vee x_3 \neq 2)$.

Definition 3.2. *Let T_c be a critical clause tree and π be a permutation. A variable node v is* dead *if its variable label comes before x in π. It is* alive *if it is not dead. All clause nodes are alive, too. A node u is* reachable *if there is a path of alive nodes from the root to u. Reachable(T_c, π) is the set of all reachable vertices. Let $G(T_c, \pi)$ be the set of clause labels of the nodes in Reachable(T_c, π) (Fig. 1).*

Lemma 3.3 (Critical clause trees model local reasoning). *Let π be a permutation of the variables and $c \in [d-1]$ a color. Let β be the restriction of α to the variables coming before x in π. Then $G(T_c, \pi) \wedge \alpha_0 \wedge \beta \models (x \neq c)$.*

We encourage the reader to verify the lemma for the critical clause tree in the figure above, for example for $\pi = (x_2, x_1, x_3, x_4)$ or $\pi = (x_1, x_2, x_3, x_4)$.

Corollary 3.4. *If $|\text{Reachable}(T_c, \pi)| \leq D$ then $c \notin \mathcal{A}(x, \alpha_0, \alpha, \pi)$. In other words, PPSZ can eliminate color c for x by local reasoning.*

Proof (Proof of Lemma 3.3). We show the following equivalent statement: Let γ be a total assignment that is compatible with $\alpha_0 \wedge \beta$ and $\gamma(x) = c \neq d$. Then γ does not satisfy $G(T_c, \pi)$. We will prove this statement constructively by finding a clause that is violated by γ.

> Set u to be the root of T_c. `do` as long as possible:
> – Let $C :=$ clause-label(u).
> – `if`   there is some $(y \neq d) \in C$ with $\gamma(y) = i \neq d$:
>   • Let v be the child of u with var-label$(v) = y$.
>   • Let w be the child of v such that edge-color$(v, w) = i$.
>   • Set $u = w$ and continue.
> – `else: return` u.

Let u be the vertex returned by this procedure. Consider any variable node v on the path from the root to u and let $y :=$ var-label(v). By construction $\beta_u(y) = \gamma(y) \neq d$. This means that y comes after x in π: Otherwise, $\gamma(y) = \alpha(y) = d$ by assumption on γ. So y comes after x, every ancestor v of u is alive, and u is reachable. Therefore $C :=$ clause-label$(u) \in G(T_c, \pi)$.

We claim that γ violates C: First consider a literal $(y \neq d) \in C$. If $\gamma(y) \neq d$, the above procedure would have continued, and not returned u. So $\gamma(y) = d$, and γ does not satisfy $(y \neq d)$. Second consider a literal $(z \neq i) \in C$ for some $i \neq d$. By Proposition 3.1 z appears as a variable label above u, and therefore $\gamma(z) = \beta_u(z)$. Since β_u violates C, it violates the literal $(z \neq i)$, thus γ violates it, too. We conclude that γ violates C. □

For $c \in [d-1]$, we define an indicator variable R_c. It is 1 if $|\text{Reachable}(T_c, \pi)| > D|$ and 0 otherwise. By the above corollary we know that $R_c = 0$ implies $c \notin \mathcal{A}(x, \alpha_0, \alpha, \pi)$. Since $d \in \mathcal{A}(x, \alpha_0, \alpha, \pi)$ for all π we get $|\mathcal{A}(x, \alpha_0, \alpha, \pi)| \leq 1 + \sum_{c=1}^{d-1} R_c$. We now have to show that $\mathbf{E}\left[\log_d\left(1 + \sum_{c=1}^{d-1} R_c\right)\right] \leq S_{d,k} + \epsilon_D$.

Note that R_c depends on the number of reachable nodes. It is difficult to understand the worst-case behavior of the random variable $\sum R_c$. Let us therefore define a new ensemble of random variables:

$$
P_c^h = \begin{cases} 1 & \text{if there exists a reachable vertex at depth } h \text{ in } T_c \text{ ,} \\ 0 & \text{else.} \end{cases}
$$

Note that if $m := |\text{Reachable}|$ is very large, then there exist a reachable vertex at depth at least h, where h is logarithmic in m. The precise connection is: Let h be the largest even integer with $2(h/2)^{(k-1)(d-1)} \leq D$. Then $R_c \leq P_c^h$. So it suffices to bound $\mathbf{E}\left[\log_d\left(1 + \sum_{c=1}^{d-1} P_c^h\right)\right]$ from above. Note that the behavior of $\sum P_c^h$ depends on (i) the shape of the critical clause trees; (ii) the concrete arrangement of variable labels in all $d - 1$ trees. All can be pretty complex. Luckily, we can prove that in the worst-case, everything looks quite nice. See the full version for a proof of the following four results.

Lemma 3.5 (Independence Between Trees, Informal). *In the worst case, the trees $T_1, \ldots, T_{d-1}$ do not share any variable labels.*

This follows from a certain monotonicity argument and the concavity of $\log_d$.

Lemma 3.6 (Independence Within a Tree, Informal). *In the worst case, no variable label appears twice within a tree.*

A version of this lemma also appears in [14]. It follows from the FKG inequality [7] and the fact that P_c^h is monotone in each of the events "y comes after x in π". At this point we can forget all about variable and clause labels. Instead of thinking of π as a permutation on $\mathcal{U}_{\alpha_0}$, we think of it as assigning each variable x a random value $\pi(x) \in [0, 1]$. With probability 1 this defines a permutation. Thus the ensemble $(P_1^h, \ldots, P_{d-1}^h)$ can be produced by the following random experiment: Select $p \in [0, 1]$ uniformly at random (this corresponds to choosing $\pi(x)$). Then delete each odd-level node with probability p, independently (if $\pi(v) < \pi(x)$ then the node labeled v is dead). Now $P_c^h = 1$ if and only if after deletion, T_c contains a path of length h starting at its root.

Observation 3.7 (Deletion in Infinite Trees, Informal). *In the worst case, all T_c are infinite trees in which an even-level node has exactly $k - 1$ children and an odd-level node exactly $d - 1$.*

This "worst case" of infinite trees can of course not happen for an actual ClSP instance F. However, it is useful to imagine infinite trees in the analysis. Let us assume the trees $T_1, \ldots, T_{d-1}$ look as in the worst case outlined above, and write $Y^h := \sum_{c=1}^{d-1} P_c^h$. The distribution of Y^h does not depend on F, only on h, d, and k. We define P_c to 1 if T_c has an *infinite* path of alive vertices and set $Y := \sum_{c=1}^{d-1} Y_c$.

Lemma 3.8. $\mathbf{E}[\log_d(1 + Y^h)]$ *converges to* $\mathbf{E}[\log_d(1 + Y)] = S_{d,k}$ *as* $h \to \infty$.

Equivalently, $\mathbf{E}[\log_d(1+Y^h)] = S_{d,k} + \epsilon_D$ for some ϵ_D that converges to 0 as D grows. To sum up,

$$\mathbf{E}_\pi[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|] \leq \mathbf{E}\left[\log_d\left(1+Y^h)\right)\right] = S_{d,k} + \epsilon_D \ .$$

4   General (d, k)-ClSP

The intuition behind the analysis of the general case is: Our partial assignment α_0 represents the current state of PPSZ (i.e. the variable assignments it has already made). If a variable x is frozen at this point in time (cf. Definition 2.4), then Lemma 2.5 gives us an upper bound on $\mathbf{E}[|\mathcal{A}(x, \alpha_0, \alpha, \pi)|]$. Otherwise, if x is not frozen, we have at least a $2/d$ chance of guessing a value for x that keeps F satisfiable.

Below we will carefully track how $\mathbf{E}[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|]$ changes over time after x becomes frozen. Surprisingly we only use one property of our D-implication mechanism: adding more information to α_0 can only decrease the number of eligible colors:

Let $y \neq x$ and $c := \alpha(y)$. Then $\mathcal{A}(x, \alpha_0 \wedge y = c) \subseteq \mathcal{A}(x, \alpha_0)$.

4.1   Definitions and Notation

Through most of the analysis, we consider a certain "snapshot" of PPSZ. At this point in time it has already assigned some variables, and we represent this by the partial assignment α_0.

Definition 4.1. *Let F be a (d, k)-ClSP formula and α_0 a partial assignment. Let $x \in \mathcal{U}(\alpha_0)$.*

– $\mathcal{A}(x, \alpha_0)$ *is the set of eligible values as in Definition 2.2.*
– $\mathcal{S}_{\alpha_0}(x)$ *is the set of values $c \in [d]$ such that $F \wedge \alpha_0 \wedge (x = c)$ is satisfiable.*
– $\mathcal{S}_{\alpha_0} := \{(x, c) \in \mathcal{U}(\alpha_0) \times [d] \,\big|\, c \in \mathcal{S}_{\alpha_0}(x)\}.$

Note that a variable x is frozen if and only if $|\mathcal{S}_{\alpha_0}(x)| = 1$. Also, $\mathcal{S}_{\alpha_0}(x) \subseteq \mathcal{A}(\alpha_0, x)$. We partition the set $\mathcal{U}(\alpha_0)$ of yet unassigned variables into the parts: $\mathcal{U}(\alpha_0) = V_{\text{fo}}(\alpha_0) \,\dot\cup\, V_{\text{fr}}(\alpha_0) \,\dot\cup\, V_{\text{nf}}(\alpha_0)$ where

– $V_{\text{nf}}(\alpha_0) := \{x \in \mathcal{U}(\alpha_0) \mid |\mathcal{S}_{\alpha_0}(x)| \geq 2\}$, i.e., the set of non-frozen variables.
– $V_{\text{fo}}(\alpha_0) := \{x \in \mathcal{U}(\alpha_0) \mid |\mathcal{A}(\alpha_0, x)| = 1\}$, i.e., those variables for which the D-implication mechanism of PPSZ can rule out all but one value. Clearly, such a variable is also frozen. We call such a variable *forced*.
– $V_{\text{fr}}(\alpha_0) :=$ the set of frozen variables not in $V_{\text{fo}}(\alpha_0)$.

Lemma 2.5 guarantees that $\mathbf{E}_\pi[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|] \leq S_{d,k} + \epsilon_D$ whenever x is frozen. We write $S := S_{d,k} + \epsilon_D$ and $G := \max\{S, 1 - \frac{\log_d e}{2}\}$. As in [10] we define a *cost function*:

Definition 4.2. *Let α_0 be a partial and α a total assignment and x a variable. We define $\mathrm{cost}(\alpha_0, \alpha, x)$ as follows:*

- *If $x \notin \mathcal{U}(\alpha_0)$ or α_0, α are incompatible or α violates F, or x is forced with respect to α_0 then $\mathrm{cost}(\alpha_0, \alpha, x) = 0$;*
- *else if $x \in \mathrm{V_{nf}}(\alpha_0)$ then $\mathrm{cost}(\alpha_0, \alpha, x) = G$;*
- *else (if $x \in \mathrm{V_{fr}}(\alpha_0)$) then $\mathrm{cost}(\alpha_0, \alpha, x) = \mathbf{E}_\pi[\log_d(|\mathcal{A}(x, \alpha_0, \alpha, \pi)|)]$.*

We define $\mathrm{cost}(\alpha_0, \alpha) = \sum_{x \in \mathcal{U}(\alpha_0)} \mathrm{cost}(\alpha_0, \alpha, x)$.

Note that $\mathrm{cost}(\alpha_0, \alpha) \leq G \cdot n(\alpha_0)$ by Lemma 2.5.

4.2 A Distribution over Satisfying Assignments

Let α_0 be a partial assignment such that $F \wedge \alpha_0$ is satisfiable. We define a (not computationally efficient) process that samples a random satisfying assignment:

```
while U(α₀) ≠ ∅:
  – Pick (x, c) ∈ S_{α₀}.
  – Add (x = c) to α₀.
return α₀.
```

Note that this process always outputs a total satisfiable assignment compatible with (the original) α_0. Let $\mathrm{Q}(\alpha_0, \alpha)$ be the probability that this process, started with α_0, outputs α. This defines a probability distribution over the set of satisfying assignments of F. Let $p(\alpha_0, \alpha)$ denote the probability that $\mathrm{PPSZ}(F, \alpha_0)$ returns α.

Lemma 4.3. *Let α be a satisfying assignment, α_0 be a partial assignment compatible with α. Then $p(\alpha_0, \alpha) \geq \mathrm{Q}(\alpha_0, \alpha) \cdot d^{-\mathrm{cost}(\alpha_0, \alpha)}$.*

With this lemma in hand, we can finish the proof of Theorem 1.2.

Proof (of Theorem 1.2). Let α_0 be the empty assignment. Then

$$\Pr[\mathrm{PPSZ}(F, \alpha_0) \text{ succeeds}] = \sum_{\alpha \in \mathrm{sat}(F)} p(\alpha_0, \alpha)$$

$$\geq \sum_{\alpha \in \mathrm{sat}_V(F)} \mathrm{Q}(\alpha_0, \alpha) \cdot d^{-\mathrm{cost}(\alpha_0, \alpha)} \geq \sum_{\alpha \in \mathrm{sat}_V(F)} \mathrm{Q}(\alpha_0, \alpha) \cdot d^{-Gn} = d^{-Gn} \ .$$

□

The rest of this section is devoted to proving Lemma 4.3. We prove $p(\alpha_0, \alpha) \geq \mathrm{Q}(\alpha_0, \alpha) \cdot d^{-\mathrm{cost}(\alpha_0, \alpha)}$ by induction over $|\mathcal{U}(\alpha_0)|$, the number of variables unassigned in α_0. If α_0 is total the statement holds trivially.

For the induction step suppose α_0 is not total. PPSZ randomly picks $x \in \mathcal{U}(\alpha_0)$ and $c \in \mathcal{A}(x, \alpha_0)$, adds $(x = c)$ to α_0 and continues. For the rest of this inductive proof, the meaning of α and α_0 will not change. We thus drop

the α_0 from $\mathcal{S}_{\alpha_0}, \mathcal{S}_{\alpha_0}(x), \mathcal{A}(x,\alpha_0), \mathcal{U}_{\alpha_0}, \mathrm{V}_{\mathrm{nf}}(\alpha_0), \ldots$. We also write $\mathcal{S}, \mathcal{S}(x), \mathcal{A}(x)$ and write $s := |\mathcal{S}|, s(x) := |\mathcal{S}(x)|, a(x) := |\mathcal{A}(x)|$. Finally, since PPSZ adds $(x = c)$ to α_0, we have to look at partial assignments that extend α_0 by one variable. For this we write $\alpha_0^{x=c} := \alpha_0 \wedge (x = c)$. Most of the time we consider partial assignments that fix one additional variable x to $\alpha(x)$. We denote this by $\alpha_0^x := \alpha_0^{x=\alpha(x)}$.

Given the current partial assignment α_0, PPSZ randomly picks some $x \in \mathcal{U}$ and $c \in \mathcal{A}(x)$ and continues with $\alpha_0^{x=c}$. Thus

$$p(\alpha_0, \alpha) = \mathop{\mathbf{E}}_{x \in \mathcal{U}} \left[\mathop{\mathbf{E}}_{c \in \mathcal{A}(x)} [p(\alpha_0^{x=c}, \alpha)] \right] = \mathop{\mathbf{E}}_{x \in \mathcal{U}} \left[\frac{1}{a(x)} \cdot p(\alpha_0^{x=\alpha(x)}, \alpha) \right] ,$$

The second equality holds since $p(\alpha_0^{x=c}, \alpha) = 0$ for $c \neq \alpha(x)$. Applying the induction hypothesis to $\alpha_0^{x=\alpha(x)}$ (or α_0^x, for short):

$$p(\alpha_0, \alpha) \geq \mathop{\mathbf{E}}_{x \in \mathcal{U}} \left[\frac{Q(\alpha_0^x, \alpha) d^{-\operatorname{cost}(\alpha_0^x, \alpha)}}{a(x)} \right]$$

We could apply Jensen's inequality to the above expectation. However, the argument of $\mathbf{E}$ above includes Q, which is not necessarily very concentrated around its expectation, and Jensen's inequality does not seem to yield any usable bound. To circumvent this problem, we introduce a new probability distribution $\xi(x)$ over $\mathcal{U}(\alpha_0)$ that is proportional to $Q(\alpha_0^x, \alpha)$. Note that

$$Q(\alpha_0, \alpha) = \mathop{\mathbf{E}}_{(x,c) \in \mathcal{S}} Q(\alpha_0^{x=c}, \alpha) = \frac{1}{s} \sum_{x \in \mathcal{U}} \sum_{c \in \mathcal{S}(x)} Q(\alpha_0^{x=c}, \alpha) = \frac{1}{s} \sum_{x \in \mathcal{U}} Q(\alpha_0^x, \alpha) ,$$

and therefore $\xi(x) := \frac{Q(\alpha_0^x, \alpha)}{s \cdot Q(\alpha_0, \alpha)}$ is a probability distribution over $\mathcal{U}$. Thus,

$$\begin{aligned}
p(\alpha_0, \alpha) &\geq \mathop{\mathbf{E}}_{x \in \mathcal{U}} \left[\frac{1}{a(x)} \cdot Q(\alpha_0^x, \alpha) \cdot d^{-\operatorname{cost}(\alpha_0^x, \alpha)} \right] \\
&= \frac{s \cdot Q(\alpha_0, \alpha)}{|\mathcal{U}|} \mathop{\mathbf{E}}_{x \sim \xi} \left[\frac{d^{-\operatorname{cost}(\alpha_0^x, \alpha)}}{a(x)} \right] \\
&\geq \frac{s \cdot Q(\alpha_0, \alpha)}{|\mathcal{U}|} d^{\mathbf{E}_{x \sim \xi}[-\operatorname{cost}(\alpha_0^x, \alpha) - \log_d a(x)]}. \qquad \text{(by Jensen's)}
\end{aligned}$$

In order for our inductive prove to go through, the last expression should be at least $Q(\alpha_0, \alpha) \cdot d^{-\operatorname{cost}(\alpha_0, \alpha)}$. This happens if and only if

$$\frac{s}{|\mathcal{U}|} \cdot d^{\mathbf{E}_{x \sim \xi}[-\operatorname{cost}(\alpha_0^x, \alpha) - \log_d a(x)]} \geq d^{-\operatorname{cost}(\alpha_0, \alpha)} \iff$$

$$\log_d \frac{s}{|\mathcal{U}|} - \mathop{\mathbf{E}}_{x \sim \xi}[\operatorname{cost}(\alpha_0^x, \alpha) + \log_d a(x)] \geq -\operatorname{cost}(\alpha_0, \alpha) \iff$$

$$\mathop{\mathbf{E}}_{x \sim \xi}[\operatorname{cost}(\alpha_0, \alpha) - \operatorname{cost}(\alpha_0^x, \alpha)] - \mathop{\mathbf{E}}_{x \sim \xi}[\log_d a(x)] + \log_d \frac{s}{|\mathcal{U}|} \geq 0 . \qquad (2)$$

The proofs of the next three lemmas are quite technical and demanding but do not introduce new key ideas. They can be found in the full version of this paper.

Lemma 4.4. $\mathbf{E}_{x \sim \xi}[\text{cost}(\alpha_0, \alpha) - \text{cost}(\alpha_0^x, \alpha)] \geq \frac{1}{s}\left(G \sum_{y \in V_{\text{nf}}} s(y) + \sum_{y \in V_{\text{fr}}} \log_d a(y)\right).$

Lemma 4.5. $\mathbf{E}_{x \sim \xi}[\log_d a(x)] \leq \frac{\sum_{x \in \mathcal{U}} \log_d a(x)}{s} + \frac{\sum_{x \in V_{\text{nf}}} (s(x) - 1)}{s}$.

Lemma 4.6. $\log_d \frac{s}{|\mathcal{U}|} \geq \log_d(e) \frac{\sum_{y \in V_{\text{nf}}} (s(y) - 1)}{s}.$

Let $(*)$ denote the left-hand side of inequality (2).

$$
\begin{aligned}
s \cdot (*) \geq\ & G \sum_{y \in V_{\text{nf}}} s(y) + \sum_{y \in V_{\text{fr}}} \log_d a(y) - \sum_{x \in \mathcal{U}} \log_d a(x) - \sum_{x \in V_{\text{nf}}} (s(x) - 1) + \log_d(e) \sum_{y \in V_{\text{nf}}} (s(y) - 1) \\
=\ & \sum_{y \in V_{\text{nf}}} (Gs(y) - s(y) + 1 + \log_d(e)(s(y) - 1)) - \sum_{y \in \mathcal{U}} \log_d a(y)(1 - \mathbb{I}_{y \in V_{\text{fr}}}) \ .
\end{aligned}
$$

Note that $\log_d a(y)(1 - \mathbb{I}_{y \in V_{\text{fr}}})$ is equal to 0 if $y \in V_{\text{fr}} \cup V_{\text{fo}}$ and at most 1 if $y \in V_{\text{nf}}$. Thus it suffices to show that $\sum_{y \in V_{\text{nf}}} (Gs(y) - s(y) + \log_d(e)(s(y) - 1)) \geq 0$. We will show that every summand is non-negative for each $y \in V_{\text{nf}}$:

$$
Gs(y)s(y) + \log_d(e)(s(y) - 1) \geq 0 \quad \Leftrightarrow G \geq 1 - \log_d(e) \cdot \frac{s(y) - 1}{s(y)} \ .
$$

The last inequality holds because $\frac{s(y)-1}{s(y)} \geq 1/2$ for $y \in V_{\text{nf}}$ and $G \geq 1 - \frac{\log_d(e)}{2}$ by definition. This finishes the proof.

5   Conclusion and Open Problems

We have shown how to apply the PPSZ algorithm to (d,k)-ClSPs. In the unique case we established correlation inequalities showing that PPSZ behaves as expected. This improves the fastest known running time for Unique (d,k)-ClSP algorithm for almost all values (d,k). These results transfer to the general case for $k \geq 4$.

In our analysis of the general case we only distinguish frozen and non-frozen variables. That is, for non-frozen variables we make no difference between variables with two, three, or even d viable values. A more fine-grained analysis could give an improved result for the general case. However we do not know how to analyze the transition between different types of "non-frozen-ness".

We conjecture that the running time in the general case is no worse than in the unique case and that the current discrepancy for $k = 2, 3$ is a shortcoming of our analysis, not the algorithm.

References

1. Allender, E., Chen, S., Lou, T., Papakonstantinou, P.A., Tang, B.: Width parameterized SAT: time-space tradeoffs. Theory of Computing (to appear)
2. Beigel, R., Eppstein, D.: 3-coloring in time $O(1.3289^n)$. J. Algorithms **54**(2), 168–204 (2005)
3. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)
4. Björklund, A., Husfeldt, T.: Inclusion-exclusion algorithms for counting set partitions. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21–24, October 2006Berkeley, California, USA, Proceedings, pp. 575–582. IEEE Computer Society (2006)
5. Feder, T., Motwani, R.: Worst-case time bounds for coloring and satisfiability problems. J. Algorithms **45**(2), 192–201 (2002)
6. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. SIAM J. Comput. **28**(1), 57–104 (1998)
7. Fortuin, C.M., Kasteleyn, P.W., Ginibre, J.: Correlation inequalities on some partially ordered sets. Comm. Math. Phys. **22**, 89–103 (1971)
8. Grohe, M.: The structure of tractable constraint satisfaction problems. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 58–72. Springer, Heidelberg (2006)
9. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: Proceedings of the seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, pp. 289–298. Society for Industrial and Applied Mathematics (2006)
10. Hertli, T.: 3-SAT faster and simpler - unique-SAT bounds for PPSZ hold in general. SIAM J. Comput. **43**(2), 718–729 (2014)
11. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity. J. Comput. System Sci. **63**(4), 512–530 (2001). Special issue on FOCS 1998 (Palo Alto, CA)
12. Krokhin, A., Bulatov, A., Jeavons, P.: The complexity of constraint satisfaction: an algebraic approach. In: Kudryavtsev, V.B., Rosenberg, I.G., Goldstein, M. (eds.) Structural Theory of Automata, Semigroups, and Universal Algebra. NATO Science Series II: Mathematics, Physics and Chemistry, vol. 207, pp. 181–213. Springer, Netherlands (2005)
13. Li, L., Li, X., Liu, T., Xu, K.: From k-SAT to k-CSP: Two generalized algorithms. CoRR, abs/0801.3147 (2008)
14. Paturi, R., Pudlák, P., Saks, M.E., Zane, F.: An improved exponential-time algorithm for k-SAT. J. ACM **52**(3), 337–364 (2005). (electronic)
15. Paturi, R., Pudlák, P., Zane, F.: Satisfiability coding lemma. Chicago J. Theoret. Comput. Sci., Article 11, 19 (electronic) (1999)
16. Scheder, D.: PPZ for more than two truth values - an algorithm for constraint satisfaction problems. CoRR, abs/1010.5717 (2010)
17. Schöning, U.: A probabilistic algorithm for k-sat and constraint satisfaction problems. In: 40th Annual Symposium on Foundations of Computer Science, FOCS 1999, 17–18 October 1999, New York, pp. 410–414. IEEE Computer Society (1999)

18. Szeider, S.: On fixed-parameter tractable parameterizations of SAT. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 188–202. Springer, Heidelberg (2004)
19. Traxler, P.: The time complexity of constraint satisfaction. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 190–201. Springer, Heidelberg (2008)
20. Welzl, E.: Boolean Satisfiability - Combinatorics and Algorithms (Lecture Notes) (2005). www.inf.ethz.ch/~emo/SmallPieces/SAT.ps

Explaining Producer/Consumer Constraints

Andreas Schutt[1,2]([✉]) and Peter J. Stuckey[1,2]

[1] Decision Sciences, Data61, CSIRO, West Melbourne, Australia
{andreas.schutt,peter.stuckey}@data61.csiro.au
[2] Department of Computing and Information Systems, The University of Melbourne,
Melbourne, VIC 3010, Australia

Abstract. Resource-constrained project scheduling problems are one of the most studied scheduling problem, and constraint programming with nogood learning provides the state-of-the-art solving technology for them, at least when the aim is minimizing makespan. In this paper we examine the closely related problem of scheduling producers and consumers of discrete resources and reservoirs. Producer/consumer constraints model consumable resources, such as raw materials (*e.g.*, water) and money, in which event times relate to a production or consumption event. In this paper, we investigate what is the most appropriate language of learning: should we learn about the event times for production and consumption, or should be instead learn about the temporal relationships between events? For this reason, we explore global constraint propagators with explanation for producer/consumer constraints and contrast this with simple decomposition approaches. Experiments on resource-constrained project scheduling problems involving producer/consumer constraints show that nogood learning solvers are highly effective at these problems.

1 Introduction

One of the most-studied and well-known scheduling problem is the resource-constrained project scheduling problem (RCPSP) that comprises non-preemptive activities, scarce renewable resources, and precedence relations between pairs of activities. A schedule determines the start and end time of activities such that no resource limit is exceeded over the planning horizon and all precedence relations are satisfied. In RCPSP, an activity *consumes* some units of the renewable resource and releases or *produces* them at its ends. Renewable resources are very common and model, *e.g.*, manpower and machines, whereas non-renewable resources model, *e.g.*, money and energy, and have a fixed initial capacity, *i.e.*, an activity only consumes them. Reservoirs (also called inventories [15]) are consumable resources, for which activities can produce and/or consume resource units at their start and/or end. Normally, they have a maximal capacity or level and a minimal level requirement, *e.g.*, safety stock. They generalize renewable and non-renewable resources. The constraints that the activities

impose on the reservoir are called producer/consumer constraints. Typical examples for such a resource are fuel or water tanks, raw materials, and, even money in an investment project [15].

There is limited literature about producer/consumer constraints. Simonis and Cornelissens [22] brought it to the attention of the constraint programming (CP) community for the first time. They motivated different application scenarios and show how to model them with the global `cumulative` constraint. Barták [3] provided an overview of the intersection of industrial planning and scheduling problems which include producer/consumer constraints. Neumann and Schwindt [15] studied the properties of the feasible regions, proposing a branch-and-bound algorithm and filtered beam search heuristic for solving producer/consumer constraints in combination with generalized precedence constraints. They also created the test set `ubo`. Laborie [12] investigated the intersection of planning and scheduling involving producer/consumer constraints with generalized precedence relations. He proposed a balance constraint that reasons about the temporal relations between two events and can detect new temporal relations. He then developed an exact solution method with several dedicated search heuristics, which were combined to closed the remaining open instances in the test set `ubo`. Beldiceanu and Carlsson [5] presented the new global constraint `cumulatives`, which generalized the global `cumulative` constraint. This new constraint allowed negative heights in order to model producer/consumer constraints. Beck [4] investigated heuristics for scheduling problems involving inventories, and more recently Kinable [10] extended the balance constraints [12] to optional events.

We consider the scheduling of *production* and *consumption events* for discrete resources in reservoirs, as well as generalized precedence constraints (also called temporal constraints, minimal and maximal time lags, difference logic constraints) relating those events.

A *reservoir* is a finite pool for storing a resource, having a minimal $L_{\min}$ and maximal resource level $L_{\max}$. A *production event* adds an amount of resource to a reservoir at a certain time, the reservoir should not exceed its maximal level. A *consumption event* removes an amount of resource from a reservoir at a certain time, the reservoir should not go below its minimal level. An event x has a start or *event time* $t(x)$ and an effect or *height* $h(x)$ on the reservoir. If $h(x) < 0$ ($h(x) > 0$) then we have a consumption (production) event.

Generalized precedence constraints relate two events and are of the form: $t(x) - t(y) \leq d$ where d is integral. Note that this allows us to express, *e.g.*,

– *fixed separation of events*: $t(x) - t(y) = d$ as $t(x) - t(y) \leq d \wedge t(y) - t(x) \leq -d$,
– *x is before y* written $x \prec y$ as $t(x) - t(y) \leq -1$
– *x is no later than y* written as $x \preceq y$ as $t(x) - t(y) \leq 0$

Example 1. Consider a RCPSP with activities i and a resource of capacity R. Let s_i, e_i, d_i and r_i be the start time of i, the end time of i, the fixed duration of i and the fixed resource requirement of i on R. Then we can reformulate the problem as producer/consumer constraints with generalized precedence constraints. Each

activity i is modeled with a consumption event x_i and production event y_i for which $t(x_i) = s_i$, $h(x_i) = -r_i$, $t(y_i) = e_i$, and $h(y_i) = r_i$, and precedence constraints enforcing $t(x_i) + d_i = t(y_i)$ and the renewable resource is modeled as a reservoir with $L_{\min} = 0$, $L_{\max} = R$, and initial resource level at R. The initial resource level is modeled by a dummy production event z with $t(z) = 0$ and $h(z) = R$ where 0 is the start of the planning horizon.   □

Given the success of nogood learning on RCPSP and related scheduling problems [6,9,11,17,19–21,23], it is interesting to explore what is the best learning approach to tackle the producer/consumer problems with generalized precedence constraints, a class of problems that generalizes RCPSP and many other scheduling problems.

Usually in CP there is fairly well understood tradeoff: a propagator that infers more is worthwhile as long as the cost of the inference does not outweigh the benefits of the extra inference. With nogood learning, this becomes more complicated. A propagator with weaker inference may be better if it makes more local inferences, that are more reusable, in particular a decomposition may be advantageous since it introduces new local variables which may be worthwhile learning about. In the case of producer/consumer we have three distinct possibilities:

– a decomposition may be best since the local variables it introduces are valuable for learning
– a time bounds approach may be best since it uses native literals to the problem, and bounds can succinctly capture lots of schedules independent of what happened to earlier events,
– an approach based on ordering may be best, since ordering of events captures the essence of the constraints on reservoirs

The aim of this paper is to answer the question of which approach is best.

2 Preliminaries

2.1 Lazy Clause Generation

CP solves constraint satisfaction problems by interleaving propagation, which remove impossible values of variables from the domain, with search, which guesses values. All propagators are repeatedly executed until no change in domain is possible, then a new search decision is made. If propagation determines there is no solution then search undoes the last decision and replaces it with the opposite choice. If all variables are fixed then the system has found a solution to the problem. For more details see, *e.g.*, [18].

We assume we are solving a constraint satisfaction problem over set of variables $x \in \mathcal{V}$, each of which takes values from a given initial finite set of values or *domain* $D_{init}(x)$. The domain D keeps track of the current set of possible values $D(x)$ for a variable x. Define $D \sqsubseteq D'$ iff $D(x) \subseteq D'(x), \forall x \in \mathcal{V}$. The constraints of the problem are represented by propagators f which are functions from domains to domains which are monotonically decreasing $f(D) \sqsubseteq f(D')$

whenever $D \sqsubseteq D'$, and contracting $f(D) \sqsubseteq D$. If all values are removed from one domain of a variable x, *i.e.*, $D(x) = \emptyset$ then the constraints cannot be satisfied with the search decisions made and a failure is triggered.

We make use of CP with learning using the lazy clause generation (LCG) [16] approach. Learning keeps track of what caused changes in domain to occur, and on failure computes a *nogood* which records the reason for failure. The nogood prevents search making the same incorrect set of decisions again.

In an LCG solver integer domains are also represented using Boolean variables. Each variable x with initial domain $D_{init}(x) = [l..u]$ is represented by two sets of Boolean variables $[\![x = d]\!], l \leq d \leq u$ and $[\![x \leq d]\!], l \leq d < u$ which define which values are in $D(x)$. We use $[\![x \neq d]\!]$ as shorthand for $\neg[\![x = d]\!]$, and $[\![d \leq x]\!]$ as shorthand for $\neg[\![x \leq d - 1]\!]$. An LCG solver keeps the two representations of the domain in sync. For example if variable x has initial domain $[0..5]$ and at some later stage $D(x) = \{1, 3\}$ then the literals $[\![x \leq 3]\!], [\![x \leq 4]\!], \neg[\![x \leq 0]\!], \neg[\![x = 0]\!], \neg[\![x = 2]\!], \neg[\![x = 4]\!], \neg[\![x = 5]\!]$ will hold. Explanations are defined by clauses over this Boolean representation of the variables.

Example 2. Consider a simple constraint satisfaction problem with constraints $b \rightarrow x + 3 \leq y$, $\neg b \rightarrow y + 3 \leq x$, $b' \rightarrow y \leq 3$, $\neg b' \rightarrow x \leq 3$, with initial domains $D_{init}(b) = D_{init}(b') = \{0, 1\}$, and $D_{init}(x) = D_{init}(y) = \{0, 1, 2, 3, 4, 5, 6\}$. There is no initial propagation. Setting $[\![y = 2]\!]$ makes the first constraint propagate $D(b) = \{0\}$ with explanation $[\![y = 2]\!] \rightarrow \neg b$, then the second constraint propagates $D(x) = \{5, 6\}$ with explanation $\neg b \wedge [\![y = 2]\!] \rightarrow [\![5 \leq x]\!]$. The third constraint propagates $D(b') = \{0\}$ with explanation $[\![y = 2]\!] \rightarrow \neg b'$ and the last constraint sets $D(x) = \emptyset$, with explanation $[\![5 \leq x]\!] \wedge \neg b' \rightarrow false$. The graph of the implications is
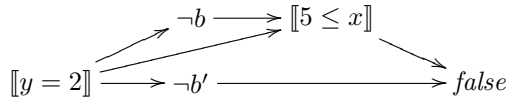
Any cut separating the decision $[\![y = 2]\!]$ from $false$ gives a nogood. The simplest one is $[\![y = 2]\!] \rightarrow false$.   □

2.2   Global Difference Logic Propagator

Constraints of difference, *i.e.*, of the form $x - y \leq d$ where d is a fixed value are one of the simplest form of constraints. Efficient algorithms based on shortest paths are known for computing satisfaction and implications, and propagation for this class of constraints. In this work we use them inside an explaining solver, hence the propagator needs to explain its reasoning, making it more akin to a difference logic theory propagator [7] in SAT modulo theories (SMT).

The global difference logic propagator [8] reasons with literals of the form $[\![x - y \leq d]\!]$, and for example can make transitive deductions like $[\![x - y \leq 1]\!] \wedge [\![y - z \leq 4]\!] \rightarrow [\![x - z \leq 5]\!]$ which are beyond the scope of usual finite domain propagation. The propagator will detect if a literal $[\![x - y \leq d]\!]$ becomes true

or false given the current set of difference constraints (and bounds) which are asserted, and also detect unsatisfiability of the set of asserted constraints. This will be important for the producer/consumer constraint which can both make use of difference logic information, and produce new difference logic information. Note that because bounds literals $[\![x \leq d]\!]$ and $[\![-x \leq d]\!]$ are so much more important in CP they are treated specially unlike in SMT, for details see [8].

3   The Producer/Consumer Constraint

Laborie [12] provides the framework of the producer/consumer (or balance) constraint for reasoning about reservoirs. In this section, we revisit his framework in order to extend it for nogood learning solvers in the next section.

Given a set of events Ev on the reservoir with $t(x)$ being the time of the event x and $h(x)$ the effect on the reservoir. Let Pr and Co be the set of producer and consumer events respectively, $Pr = \{x \in Ev \mid h(x) > 0\}$, $Co = \{x \in Ev \mid h(x) < 0\}$. The producer/consumer constraint enforces that the reservoir stays within the minimal (resource) level $L_{\min}$ and maximal (resource) level $L_{\max}$. Hence for every point in the planning horizon τ it enforces

$$L_{\min} \leq \sum_{x \in Ev : t(x) \leq \tau} h(x) \leq L_{\max}.$$

The event time $t(x)$ and height $h(x)$ can be variable. We use the notation $t_{\min}(x)$ $(t_{\max}(x))$ to refer to the current minimum (resp. maximum) possible time for event x, and similarly $h_{\min}(x)$ $(h_{\max}(x))$ for minimum (maximum) height.

In order to propagate the constraint, the set of events are partitioned into the following six sets with respect to an event x.

before	$B(x) = \{y \in Ev \mid t(y) < t(x)\}$
before or with	$BS(x) = \{y \in Ev \mid t(y) \leq t(x) \wedge y \notin B(x) \cup S(x)\}$
with	$S(x) = \{y \in Ev \mid t(y) = t(x) \vee (t(y) \leq t(x) \wedge t(y) \geq t(x))\}$
after or with	$AS(x) = \{y \in Ev \mid t(y) \geq t(x) \wedge y \notin A(x) \cup S(x)\}$
after	$A(x) = \{y \in Ev \mid t(y) > t(x)\}$
unknown	$U(x) = \{y \in Ev \mid y \notin B(x) \cup BS(x) \cup S(x) \cup A(x) \cup AS(x)\}$

Since normally not all relationships between pairs of events are known in advanced, the partition of events dynamically changes during the search. However, only events in one of these sets $BS(x)$, $AS(x)$, or $U(x)$ can move to another set and that at most twice. At the end, all events are partitioned by $B(x) \cup S(x) \cup A(x)$. An event in $U(x)$ can move to all other sets. An event in $BS(x)$ can move to either $B(x)$ or $S(x)$. An event in $AS(x)$ can move to either $S(x)$ or $A(x)$.

Determining to which set an event y belongs to during search can be performed by checking the earliest and latest event time or using ordering constraints. If we use event time information then the sets are determined by:

$$B(x) = \{y \in Ev \backslash \{x\} \mid t_{\max}(y) < t_{\min}(x)\}$$
$$BS(x) = \{y \in Ev \backslash \{x\} \mid t_{\max}(y) \le t_{\min}(x) \wedge (t_{\min}(y) < t_{\max}(y) \vee t_{\min}(x) < t_{\max}(x))\}$$
$$S(x) = \{y \in Ev \backslash \{x\} \mid t_{\max}(y) = t_{\min}(x) \wedge t_{\min}(y) = t_{\max}(y) \wedge t_{\min}(x) = t_{\max}(x)\} \cup \{x\}$$
$$AS(x) = \{y \in Ev \backslash \{x\} \mid t_{\min}(y) \ge t_{\max}(x) \wedge (t_{\min}(y) < t_{\max}(y) \vee t_{\min}(x) < t_{\max}(x))\}$$
$$A(x) = \{y \in Ev \backslash \{x\} \mid t_{\min}(y) > t_{\max}(x)\}$$
$$U(x) = \{y \in Ev \backslash \{x\} \mid t_{\min}(y) < t_{\max}(x) \wedge t_{\min}(x) \le t_{\max}(y)\}$$

To make use of ordering constraints, we first introduce the ordering Booleans B_{xy} for all event pairs $\{x, y\} \subseteq Ev$, and the constraints

$$b_{xy} \leftrightarrow t(x) < t(y) \qquad b_{yx} \leftrightarrow t(y) < t(x) \qquad \neg b_{xy} \vee \neg b_{yx}$$

unless we use the global difference logic propagator where instead we simply define the ordering literals b_{xy} as

$$b_{xy} \equiv [\![t(x) - t(y) \le -1]\!] \qquad b_{yx} \equiv [\![t(y) - t(x) \le -1]\!]$$

Using the ordering literals the sets are determined by:.

$$B(x) = \{y \in Ev \backslash \{x\} \mid b_{xy} = true\}$$
$$BS(x) = \{y \in Ev \backslash \{x\} \mid D(b_{xy}) = \{true, false\} \wedge b_{yx} = false\}$$
$$S(x) = \{y \in Ev \backslash \{x\} \mid b_{xy} = false \wedge b_{yx} = false\} \cup \{x\}$$
$$AS(x) = \{y \in Ev \backslash \{x\} \mid b_{xy} = false \wedge D(b_{yx}) = \{true, false\}\}$$
$$A(x) = \{y \in Ev \backslash \{x\} \mid b_{yx} = true\}$$
$$U(x) = \{y \in Ev \backslash \{x\} \mid D(b_{xy}) = \{true, false\} \wedge D(b_{yx}) = \{true, false\}\}$$

Note that the relationships determined by the ordering constraints will be strictly more informative than those determined from event time information (independent of whether we use difference logic or not). The conditions for $B(x)$, $S(x)$ and $A(x)$ for event time, will enforce the conditions for ordering literals. Thus, the ordering literals can potentially decide earlier to which set an event y belongs to and exploit that information for propagation.

3.1   Immediate Maximal Resource Level Before Event

In this sub-section following [12], we describe the consistency check and the filtering on the event times and heights. The maximal resource level shortly before an event $x \in Ev$ can be approximated by

$$L_{\max}^{<}(x) = \sum_{y \in B(x)} h_{\max}(y) + \sum_{y \in Pr \cap (BS(x) \cup U(x))} h_{\max}(y) \qquad (1)$$

where the last sum is an approximation because it neglects precedence relations and consumer events.

Consistency. If the maximal resource level is too low then there does not exist any producer event that can be pushed for execution before the event x. Thus, the system is inconsistent, i.e.,

$$L_{\max}^{<}(x) < L_{\min} \rightarrow false.$$

Time Bounds Filtering. If the first sum in (1) is less than $L_{\min}$ then the lower bound on $t(x)$ can be improved.

$$\sum_{y \in B(x)} h_{\max}(y) < L_{\min} \rightarrow \exists \Omega \subseteq Pr \cap (BS(x) \cup U(x)) \text{ with}$$

$$\sum_{y \in B(x) \cup \Omega} h_{\max}(y) \geq L_{\min} : \forall y \in \Omega : t(y') < t(x)$$

Searching for an arbitrary set Ω is expensive, but a "practical" set can be computed as follows. Let the events $y_1, y_2, \ldots \in Pr \cap (BS(x) \cup U(x))$ be in chronological order according to their earliest event time, *i.e.*, $t_{\min}(y_1) \leq t_{\min}(y_2) \leq \ldots$. Then the lower bound on $t(x)$ can be updated as follows.

$$\exists k : \sum_{i=1}^{k-1} h_{\max}(y_i) < L_{\min} - \sum_{y \in B(x)} h_{\max}(y) \leq \sum_{i=1}^{k} h_{\max}(y_i) \rightarrow t_{\min}(y_k) < t(x)$$

Consumption and Production Level Filtering. The consumption or production level of certain events y can be filtered with the respect to x. The following rule holds for all consumers $y \in Co \cap B(x)$ and all producers $y \in Pr \cap (B(x) \cup BS(x) \cup U(x))$.

$$L_{\max}^{<}(x) + h_{\min}(y) - h_{\max}(y) < L_{\min} \rightarrow L_{\min} - L_{\max}^{<}(x) + h_{\max}(y) \leq h(y)$$

This rules avoids a resource level that is below the safety level $L_{\min}$.

3.2   Other Resource Levels Regarding an Event

Similar reasoning to the immediate maximal resource level before an event can be performed for the immediate minimal resource level before the event (2) and the immediate maximal (3) and minimal resource level after the event (4). They lead to similar propagations to those described for $L_{\max}^{<}(x)$ in the previous subsection.

$$L_{\min}^{<}(x) = \sum_{y \in B(x)} h_{\min}(y) + \sum_{y \in Co \cap (BS(x) \cup U(x))} h_{\min}(y) \tag{2}$$

$$L_{\max}^{>}(x) = \sum_{y \in B(x) \cup BS(x) \cup S(x)} h_{\max}(y) + \sum_{y \in Pr \cap (AS(x) \cup U(x))} h_{\max}(y) \tag{3}$$

$$L_{\min}^{>}(x) = \sum_{y \in B(x) \cup BS(x) \cup S(x)} h_{\min}(y) + \sum_{y \in Co \cap (AS(x) \cup U(x))} h_{\min}(y) \tag{4}$$

4   Explanations

In this section, we describe how to explain propagations for $L_{\max}^<$, the propagations for other resource levels are explained similarly. Since we investigate different propagation algorithms, we introduce general explanations at first before refining them to each of the propagators.

4.1   Explanation of the Resource Level and Inconsistency

Explanation for inconsistencies or filtering regarding to an event x must express the reason for the current bound on the resource level $L_{\max}^<(x)$. For simplicity, we assume fixed consumption/production of the events at first. In this case, the bound on the level is caused by the following reasons.

– consumer events are executed before event x
– producer events are executed simultaneously to or after event x

All those events cause a lower level on the reservoir shortly before event x. Thus, the corresponding explanation would be

$$expl(L_{\max}^<(x)) \;=\; \bigwedge_{y \in Co \cap B(x)} expl(y \;\prec\; x) \;\wedge\; \bigwedge_{y \in Pr \cap Z(x):y \neq x} expl(y \;\succeq\; x) \quad (5)$$

where $Z(x) = S(x) \cup AS(x) \cup A(x)$ and the functions $expl(y \prec x)$ and $expl(y \succeq x)$ are defined later. Note that the relative position of events that are not considered are irrelevant for the current bound on the level. Thus, they can be left out of the explanation.

In the case, events can have flexible consumption/production then the bound on the resource level can be caused by these additional reasons.

– consumer events run before event x consume too many resource units
– production events that are not simultaneously to or after event x produce too few resource units

Thus, the explanation (5) must be extended by

$$\bigwedge_{y \in Co \cap B(x)} [\![h(y) \le h_{\max}(y)]\!] \;\wedge\; \bigwedge_{y \in Pr \cap (B(x) \cup BS(x) \cup U(x))} [\![h(y) \le h_{\max}(y)]\!]$$

Explanation for Inconsistency. If we have resource underflow $L_{\max}^<(x) < L_{\min}$ immediately before an event x then the explanation is simply

$$expl(L_{\max}^<(x)) \;\to\; false.$$

We generalize this explanation by removal of consumer and producer events considered in (5) in input order using the slack $L_{\min} - L_{\max}^<(x) - 1$.

4.2   Explanation for Time Bounds Filtering

The lower bound on the event time of x can be updated if all events that are before x are not enough to achieve the minimal resource level. In that case, some producer events that are currently in the before-or-simultaneously ($BS(x)$) or unknown relationship ($U(x)$) to x, must happen before x in any solution. In addition to the reasons considered for the current bound on the maximal resource level, these reasons need to be considered for a bound update

– producer events in $BS(x) \cup U(x)$ that start too late

These reasons result in the following explanation.

$$expl(L^{<}_{\max}(x)) \wedge \bigwedge_{i=k}^{\infty} [\![t_{\min}(y_k) \le t(y_i)]\!] \to t_{\min}(y_k) < t(x)$$

We generalize this explanation in the similar way as in the case of inconsistency, but we use the slack $\sum_{i=1}^{k} h_{\max}(y_i) - L_{\min}$. At first, we remove events in $B(x)$ in input order and then the remaining events in chronological order of their minimal event time.

4.3   Explanation for Consumption and Production Level Filtering

The lower bound on the height of an event y can be updated if it would cause a resource underflow shortly before event x. The potential underflow is only related the current bound on the maximal level and an explanation is simply

$$expl(L^{<}_{\max}(x)) \to L_{\min} - L^{<}_{\max}(x) + h_{\max}(y) \le h(y).$$

5   Global Reservoir Propagators

In this section, we describe three different reservoir propagators and specialize the general explanations described in the previous section.

5.1   Bounds Propagator

The bounds propagator `bounds` uses the current bounds on the event times in order to relate pairs of events, *i.e.*, partitioned all events into the six sets described in the previous section with respect to an event x. For each event, the propagator can easily determine the event partition in linear time by scanning over all events. Thus, it can also determine all four reservoir levels and time for the time bounds filtering in linear time, for the last one we assume that the events are sorted with respect to $t_{\min}(.)$. The sorting can be done at the beginning of the propagator's execution. Therefore, the worst case complexity of the bounds propagator is $\mathcal{O}(|Ev|^2)$.

When the propagator `bounds` detects a inconsistency or performs filtering it additional generates the explanation described in the previous section and specializes them by using following bounds on the event times.

$$expl(y \prec x) = [\![t(y) < t_{\min}(x)]\!] \wedge [\![t_{\min}(x) \le t(x)]\!]$$
$$expl(y \succeq x) = [\![t_{\max}(x) \le t(y)]\!] \wedge [\![t(x) \le t_{\max}(x)]\!]$$

Note the literals $[\![t_{\min}(x) \le t(x)]\!]$ and $[\![t(x) \le t_{\max}(x)]\!]$ will only appear once in an explanation. Since most of the explanation, e.g. $expl(L_{\max}^<(x))$, can be pre-computed in linear time for an event x. It does not add to the worst case complexity of the propagator.

5.2   Order Propagator

The order propagator `order` works in the same way as `bounds`, but it uses ordering variables between pairs of events for propagation. Hence, it has the same runtime complexity $\mathcal{O}(|Ev|^2)$. It uses ordering literals in the explanations.

$$expl(y \prec x) = b_{yx} \qquad\qquad expl(y \succeq x) = \neg b_{yx}$$

Note that Laborie [12] uses the non-learning version of the propagator `order`.

5.3   Timetable Propagator

The timetable propagator `tt` does not consider the time relations between pair of events, but relates events to time points in the planning horizon. Instead of performing propagation at each time point in the horizon, it only has to consider time points at which the minimal or maximal level may changes. These time points are the bounds on the event times, *i.e.*, $t_{\min}(x)$ and $t_{\max}(x)$. Hence, it considers a number of time points which is linear in the number of events.

The propagation and explanation work similar to the `bounds` propagator. Consider the time point $\tau = t_{\max}(x)$ of an event x for propagation and let z be an *artificial event* with $t(z) = t_{\min}(z) = t_{\max}(z) = \tau$ and $h(z) = 0$ then the same propagation can be done as the `bounds` propagator with respect to this fixed artificial event. Consequently, the explanation are the same and reduce to one of the following literals due to the fixed event z.

$$expl(y \prec z) = [\![t(y) < \tau]\!] \qquad\qquad expl(y \succeq z) = [\![\tau \le t(y)]\!]$$

It is obvious that the `tt` propagator can easily be implemented with a worst case complexity $\mathcal{O}(|Ev|^2)$. Note that this propagator is a specialized form of the timetable propagator of the global `cumulative` constraint. Thus, it will prune the same as the timetable propagator in `cumulative` when the reservoir constraints are modeled as cumulative propagators as described in [22].

6   Models

We use the solver-independent modeling language MiniZinc [13] for describing
our models. The input parameter of an instance are as follows. Their meaning
is giving in the comment after the declaration.

```
set of int: R;   % Set of reservoirs
set of int: E;   % Set of events
set of int: P;   % Set of generalized precedence relations
array [R] of int: Lmin;   % Minimum level of the reservoirs
array [R] of int: Lmax;   % Maximum level of the reservoirs
array [R, E] of int: rr;    % Amount of resource production/consumption of
    the events
array [P, 1..3] of int: prec;    % Generalized precedence relations between
    two events: start(prec[i,1]) + prec[i,2] <= start(prec[i,3])
array [R] of set of int: resE = [ {i | i in E where rr[r,i] != 0} | r in R
    ];   % Set of"non-zero" events for each reservoir
```

Then the start (event) time variables `s` and the objective `objective` are
declared as follows where `0..UB` is the planning horizon.

```
set of int: Times = 0..UB;      % Planning horizon
array [E] of var Times: s;      % Event time variables
var Times: objective = s[sink]; % Objective variable
```

The initial upper bound on the objective `UB` is initialized by `sum(i in
E)(max([0] ++ [prec[p,2] | p in P where prec[p,1] = i]))` unless oth-
erwise stated. Note that the set of events `E` contains one dummy source (`source`)
and sink (`sink`) event. The source start time is constrained by `s[source] = 0`.
Generalized precedence constraints are expressed by one binary linear inequality
constraints as usual.

```
constraint forall(p in P)( s[prec[p, 1]] + prec[p, 2] <= s[prec[p, 3]] );
```

Reservoir Decompositions. The first decomposition is the time-indexed formu-
lation (`dTT`), which allows learning about the event times. This model creates
two linear constraints for each point in time in the planning horizon. Thus the
model size is time-dependent.

```
array[E, Times] of var bool: bit =
    array2d(E, Times, [s[i] <= t | i in E, t in Times]);
constraint forall(r in R, t in Times)(
    sum(i in res_events[r])(rr[r,i] * bit[i,t]) <= Lmax[r]
/\  sum(i in res_events[r])(rr[r,i] * bit[i,t]) >= Lmin[r] );
```

Note that the additional Boolean variables `bit` are literals of the Boolean repre-
sentation of `s` in LCG solver, *i.e.*, those already exists for LCG solvers. In total
$\mathcal{O}(|R| \cdot UB)$ linear constraints of size $\mathcal{O}(|E|)$ are created.

The second decomposition is an event-based formulation (`dAfter`), which
ensures that the resource levels are within the limits at the event time for each
event. For modeling it, we required two additional sets of Boolean variables
`s_eq_0` and `s_leq_s`, where the first set describes whether an event starts at
time point 0 and the second set are order variables between pairs of events.

```
array[E] of var bool: s_eq_0  = [s[i] <= 0 | i in E];
array[E,E] of var bool: s_leq_s = array2d(E, E, [s[i] <= s[j] | i, j in E])
    ;
% Constraints for correct reservoir levels at event time
constraint forall(r in R, i in resE[r])(
        sum(j in resE[r])( rr[r,j] * s_leq_s[j,i] ) <= Lmax[r]
    /\  sum(j in resE[r])( rr[r,j] * s_leq_s[j,i] ) >= Lmin[r] );
% Constraints for correct reservoir level at time point 0
constraint forall(r in R)(
    if Lmin[r] > 0
    then sum(i in res_events[r])(rr[r,i] * s_eq_0[i]) >= Lmin[r]
    else if Lmax[r] < 0
        then sum(i in res_events[r])(rr[r,i] * s_eq_0[i]) <= Lmax[r]
    else true endif endif);
% Constraints for s_leq_s
constraint forall(i, j in E where i < j)(s_leq_s[i, j] \/ s_leq_s[j, i]);
```

The first set of constraints enforces the reservoir level at each event time, whereas the second enforces it for time point 0, and the last set of constraints explicitly models that one of the pair of order variables must be true. Due to the order variables `s_leq_s`, a learning solver is able to learn about the temporal relations between pairs of events. The reified constraints `s_leq_s[i,j] <-> s[i] <= s[j]` are created in the definition of `s_leq_s`. The decomposition (`after`) creates $\mathcal{O}(|R| \cdot |E|)$ linear constraints of size $\mathcal{O}(|E|)$ and $\mathcal{O}(|E|^2)$ Boolean variables and reified binary linear constraints.

The third decomposition (`dBefore`) is an extension of the second one (`dAfter`). It adds redundant linear constraints for the reservoir level shortly before the event time for each event. It re-uses the Boolean variables `s_eq_0` and `s_leq_s`.

```
constraint forall(i in resE[r])(
    sum(j in resE[r])(rr[r,j]*not(s_leq_s[i,j])) <= Lmax[r]-Lmin[r]*s_eq_0[i
        ]
/\ sum(j in resE[r])(rr[r,j]*not(s_leq_s[i,j]))-Lmin[r]*not(s_eq_0[i]) >=
    0);
```

In addition to the constraints and variables created by the decomposition (`dAfter`), it creates $\mathcal{O}(|R| \cdot |E|)$ linear constraints of size $\mathcal{O}(|E|)$.

Global Reservoir Constraints. Rather than using decompositions to model the reservoir constraints we can make use of the global propagators. The `tt` propagators has the same propagation strength as the decomposition `dTT`, but drastically reduces the model size and makes it time independent, but note it does not have the same *language of learning*. The `order` propagator is equivalent to the `dBefore` decomposition, but again drastically smaller in size. Again it does not have many intermediate variables which might be useful for learning. The `bounds` propagator is weaker than the `dBefore` and `dAfter` decompositions, since it does not reason about order, but it is, of course, concise, and avoids the need for any order variables.

7   Experiments

The experiments were run on a machine running Ubuntu 14.04 with an Intel(R) Core(TM) i7 CPU running at 2.8 GHz with 8 GB memory. We implemented the

Table 1. Results on `ubo50`

Solver	#inst	#opt	#sat	#unk	#inf	m.rt	m.it	m.#confl	m.#nodes
`neu&sch`	90	47	1	0	42	16.23s	0.0s		
`order+diff`	90	48	0	0	42	0.07s	0.01s	48	757
`order`	90	48	0	0	42	0.33s	0.01s	305	1823
`bounds+diff`	90	48	0	0	42	0.50s	0.01s	2143	6517
`bounds`	90	48	0	0	42	0.56s	0.01s	2384	6741
`tt+diff`	90	48	0	0	42	0.94s	0.01s	314	2131
`tt`	90	48	0	0	42	0.86s	0.01s	282	2108
`dTT`	90	48	0	0	42	1.21s	0.71s	259	735
`dAfter`	90	48	0	0	42	0.35s	0.08s	330	2115
`dBefore`	90	48	0	0	42	0.53s	0.14s	346	2101

Table 2. Results on `ubo100`

Solver	#inst	#opt	#sat	#unk	#inf	m.rt	m.it	m.#confl	m.#nodes
`neu&sch`	90	50	7	2	31	60.11s	0.0s		
`order+diff`	90	57	0	0	33	0.89s	0.03s	229	2818
`order`	90	55	2	0	33	18.16s	0.03s	3996	17758
`bounds+diff`	90	55	2	0	33	20.55s	0.02s	38048	103067
`bounds`	90	55	2	0	33	22.02s	0.02s	44001	122138
`tt+diff`	90	55	2	1	32	31.96s	0.02s	6718	32048
`tt`	90	55	2	1	32	33.61s	0.02s	6655	33665
`dTT`	90	57	0	1	32	17.23s	3.45s	2045	12001
`dAfter`	90	56	1	0	33	11.41s	0.30s	1800	12509
`dBefore`	90	56	1	0	33	18.22s	0.57s	2347	14017

described propagators in the lazy clause generation solver Chuffed (`chuffed`). We ran all model with a smallest first search alternating on each restart with activity based search. We used the benchmarks `ubo10`, `ubo20`, `ubo50`, and `ubo100` created by [15] consisting of 90 instances with 10, 20, 50, and 100 events, respectively. Since all instances are closed, we constructed the new benchmark `ubo200` [1] consisting of 90 instances with 200 events in the same way as in [15].

While the main purpose of our experiments is to determine the best form of explanation for this class of problem, in order to calibrate the learning methods against other approaches we also ran the method of Neumann and Schwindt [15] (`neu&sch`) on a machine running Windows 10 and having a Intel(R) Core(TM) i5 CPU with 3.2 GHz and 4 GB memory (since we only have a Windows

---

[1] Available at http://people.eng.unimelb.edu.au/pstuckey/rcpsp/.

executable). We were unable to obtain an executable of Laborie's method [12] to compare with. All runs were limited to 10 min.

We compare the following variations of decompositions and global propagators: (`dTT`) `dTT` decomposition, (`dAfter`) `dAfter` decomposition, (`dBefore`) `dBefore` decomposition, (`order`) `order` propagator, (`bounds`) `bounds` propagator, (`tt`) `tt` propagator, (`order+diff`) `order` and `diff` propagator, (`bounds+diff`) `bounds` and `diff` propagator, and (`tt+diff`) `tt` and `diff` propagator.

7.1   Results

The results are shown in Table 1, 2 and 3. Each table shows: (#inst) the number of benchmark instances, (#opt) the number of instances proved optimal in the time limit, (#sat) the number where a solution was found, but was not proven optimal in the time limit, (#unk) the number where no solution was found, (#inf) the number proved unsatisfiable, (m.rt) the mean runtime in seconds, (m.it) the mean initialization time in seconds, (m.#confl) the mean number of conflicts, and (m.#nodes) the mean number of nodes. If the solver timed out for an instance then the number of conflicts (nodes) at that time are counted in m.#confl (m.#nodes).

Table 3. Results on `ubo200`

Solver	#inst	#opt	#sat	#unk	#inf	m.rt	m.it	m.#confl	m.#nodes
`neu&sch`	90	54	10	11	15	148.92s	0.0s		
`order+diff`	90	66	0	0	24	24.67s	0.09s	2056	22454
`order`	90	53	12	4	21	154.49s	0.09s	6676	29181
`bounds+diff`	90	60	5	4	21	82.70s	0.05s	75348	245707
`bounds`	90	61	4	4	21	83.86s	0.05s	75297	242386
`tt+diff`	90	52	13	4	21	160.49s	0.05s	4498	21846
`tt`	90	51	14	5	20	166.96s	0.05s	4644	23048
`dTT`	90	0	0	90	0	—	∞	—	—
`dAfter`	90	52	14	3	21	155.67s	1.61s	4636	24930
`dBefore`	90	51	15	2	22	172.75s	3.03s	4634	21900

Note that learning was crucial for solving these problems, even on smaller benchmarks (`ubo20`) the best method `order+diff` was unable to solve all problems within the 10 min time limit without learning, whereas with learning all our methods could easily solve all these problems, and the larger ones (`ubo50`).

The method of Neumann and Schwindt is not competitive, for the smallest sizes it can solve most problems, but significantly slower, and it is very poor at detecting infeasible problems. The order based explanations are clearly dominant in terms of number of conflicts and number of nodes. The difference logic

propagator, which clearly supports the order based explanations is significantly advantageous to the point where in Table 3 even though order based search without the difference logic propagator is massively smaller than linear explanation search, it cannot prove optimality of as many instances.

What is surprising is how effective the decompositions are, at least in terms of search. Clearly the intermediate literals introduced by the decompositions are improving the learning. The decompositions are actually better than all methods other than the order-based globals until size 200 where the size of the decompositions become disadvantageous. The time decomposition initialization time grows quickly, to the point that for `ubo200` it prevents the method running.

As the size grows the tradeoff of weaker propagation but better runtime complexity the `bounds` global propagator (`bounds,bounds+diff`) over the `tt` global propagator (`tt,tt+diff`) becomes evident. It still cannot compete with the `order` propagator, when used in conjunction with globals difference propagation. This may well change with incremental versions of these propagators.

We also examined the 12 hard instances of the benchmark set generated by Neumann *et al.* [14] and closed by Laborie [12] using ILOG Scheduler. Table 4 compares with the published results of Neumann *et al.* [15] and Laborie [12] on a HP-UX 9000/785 workstation. Note that Laborie only presents the best results from his nine different heuristics for each instance.

Table 4. Comparison with Neumann et al. [15] and Laborie [12].

Instance	Optimal	`neu&sch`	Best of `laborie`	`order+diff`
50_10	92	time out	0.28 s	0.03 s
50_27	96	346.483 s	2.43 s	0.1s
50_82	Unsat	509.161 s	0.05 s	0.03 s
100_6	211	time out	0.97 s	2.3 s
100_12	197	time out	0.72 s	0.88 s
100_20	199	time out	0.46 s	13.6 s
100_30	204	time out	2.11 s	0.85 s
100_41	337	time out	0.62 s	1.62 s
100_43	Unsat	time out	7.65 s	0.54 s
100_54	344	time out	0.46 s	6.95 s
100_58	317	time out	0.49 s	0.37 s
100_69	Unsat	time out	1.96 s	0.41 s

Clearly `order+diff` is comparable to the best of Laborie's 9 method (although his CPU is somewhat slower). His approaches includes the inference of new ordering relationships, and specialized search routines that make use of the information about precedences inferred by the search. ILOG scheduler includes a component equivalent to the global difference logic propagator, without learning.

As the combinatorics of these problems grow substantially as the size increases we are confident that our learning solver would outperform his approach on larger problems, although perhaps we would need to invest in incremental propagators and inference of precedences to match the well engineered commercial solver.

8   Conclusion

The producer/consumer constraints is a powerful tool for specifying complex scheduling problems with renewable and non-renewable resources. In this paper we have explored what is right approach to solving these problems once we are using nogood learning. The key language of learning is the ordering literals b_{xy} used by the `order` propagator, *but with the proviso* that we need to use a global difference logic propagator to see the interaction of the inference between ordering literals. Surprisingly without the use of the global difference logic propagator, decompositions based on ordering are competitive, since they generates many intermediate literals which prove to be useful for learning, even if the models they create are far larger.

There is considerable scope for improving the producer/consumer propagators with learning. Making the propagators incremental, and extending the order propagator to create new order inferences are likely to be highly beneficial. Given the effectiveness of the decompositions, at least in terms of search, it might be worth investigating a global propagator that supports lazy decomposition [1,2] where intermediates are made available during search in parts of the global where many explanations are generated.

References

1. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Stuckey, P.J.: To encode or to propagate? The best choice for each constraint in SAT. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 97–106. Springer, Heidelberg (2013)
2. Abío, I., Stuckey, P.J.: Conflict directed lazy decomposition. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 70–85. Springer, Heidelberg (2012)
3. Barták, R.: Conceptual models for combined planning and scheduling. Electron. Notes Discr. Math. **4**, 1 (2000)
4. Beck, J.C.: Heuristics for scheduling with inventory: dynamic focus via constraint criticality. J. Sched. **5**(1), 43–69 (2002)
5. Beldiceanu, N., Carlsson, M.: A new multi-resource cumulatives constraint with negative heights. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 63–79. Springer, Heidelberg (2002)

6. Berthold, T., Heinz, S., Lübbecke, M.E., Möhring, R.H., Schulz, J.: A constraint integer programming approach for resource-constrained project scheduling. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 313–317. Springer, Heidelberg (2010)

7. Cotton, S., Maler, O.: Fast and flexible difference constraint propagation for DPLL(T). In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 170–183. Springer, Heidelberg (2006)

8. Feydy, T., Schutt, A., Stuckey, P.J.: Global difference constraint propagation for finite domain solvers. In: Antoy, S. (ed.) Proceedings of 10th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, pp. 226–235. ACM Press (2008)

9. Horbach, A.: A boolean satisfiability approach to the resource-constrained project scheduling problem. Ann. Oper. Res. **181**(1), 89–107 (2010)

10. Kinable, J.: A reservoir balancing constraint with applications to bike-sharing. In: Quimper, C.-G., Cavallo, M. (eds.) CPAIOR 2016. LNCS, vol. 9676, pp. 216–228. Springer, Heidelberg (2016). doi:10.1007/978-3-319-33954-2_16

11. Kreter, S., Schutt, A., Stuckey, P.J.: Modeling and solving project scheduling with calendars. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 262–278. Springer, Heidelberg (2015)

12. Laborie, P.: Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. Artif. Intell. **143**(2), 151–188 (2003)

13. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.R.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)

14. Neumann, K., Zimmermann, J.: Methods for resource-constrained project scheduling with regular and nonregular objective functions and schedule-dependent time windows. In: Węglarz, J. (ed.) Project Scheduling: Recent Models, Algorithms and Applications, vol. 14, pp. 261–287. Springer, New York (1999)

15. Neumann, K., Schwindt, C.: Project scheduling with inventory constraints. Math. Methods Oper. Res. **56**(3), 513–533 (2002)

16. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)

17. Schnell, A., Hartl, R.F.: On the efficient modeling and solution of the multi-mode resource-constrained project scheduling problem with generalized precedence relations. OR Spectr. **38**(2), 283–303 (2015)

18. Schulte, C., Stuckey, P.J.: Efficient constraint propagation engines. ACM Trans.Program. Lang. Syst. **31**(1), 2 (2008)

19. Schutt, A., Chu, G., Stuckey, P.J., Wallace, M.G.: Maximising the net present value for resource-constrained project scheduling. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 362–378. Springer, Heidelberg (2012)

20. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints **16**(3), 250–282 (2011)

21. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving RCPSP/max by lazy clause generation. J. Sched. **16**(3), 273–289 (2013)

22. Simonis, H., Cornelissens, T.: Modelling producer/consumer constraints. In: Montanari, U., Rossi, F. (eds.) Principles and Practice of Constraint Programming - CP '95. LNCS, vol. 976, pp. 449–462. Springer, Heidelberg (1995)

23. Szeredi, R., Schutt, A.: Modelling and solving multi-mode resource-constrained project scheduling. In: Rueher, M. (ed.) CP 2016. LNCS, vol. 9832, pp. 483–492. Springer, Heidelberg (2016)

Learning from Learning Solvers

Maxim Shishmarev[1(✉)], Christopher Mears[1,2], Guido Tack[1,2],
and Maria Garcia de la Banda[1,2]

[1] Faculty of IT, Monash University, Melbourne, Australia
{maxim.shishmarev,chris.mears,guido.tack,
maria.garciadelabanda}@monash.edu
[2] Data61/CSIRO, Melbourne, Australia

Abstract. Modern constraint programming solvers incorporate SAT-style clause learning, where sets of domain restrictions that lead to failure are recorded as new clausal propagators. While this can yield dramatic reductions in search, there are also cases where clause learning does not improve or even hinders performance. Unfortunately, the reasons for these differences in behaviour are not well understood in practice. We aim to cast some light on the practical behaviour of learning solvers by profiling their execution. In particular, we instrument the learning solver *Chuffed* to produce a detailed record of its execution and extend a graphical profiling tool to appropriately display this information. Further, this profiler enables users to measure the impact of the learnt clauses by comparing Chuffed's execution with that of a non-learning solver, and examining the points at which their behaviours diverge. We show that analysing a solver's execution in this way can be useful not only to better understand its behaviour — opening what is typically a black box — but also to infer modifications to the original constraint model that can improve the performance of both learning and non-learning solvers.

1 Introduction

Lazy Clause Generation (LCG) [5,10] is a powerful solving technique that combines the strengths of Constraint Programming and SAT solving. It works by instrumenting finite domain propagation to record the reasons for each propagation step, thus creating an implication graph like the ones built by a SAT solver [7]. This graph is used to derive nogoods (i.e., reasons for failure) which can be recorded as clausal propagators and propagated efficiently using SAT technology. The combination of constraint propagation and clause learning can dramatically reduce search and greatly improve performance.

Indeed, LCG solvers are the state of the art for solving a number of hard combinatorial optimisation problems, such as Resource Constrained Project Scheduling Problems [12] and the Carpet Cutting Problem [13]. Further, they consistently exhibit better performance than traditional Constraint Programming (CP) solvers for a large number of problems in the annual MiniZinc Challenge [15]. Yet, for some problems, LCG solvers seem to be unable to benefit

---

from the learnt clauses and perform poorly compared to non-learning competitors. The reasons for these differences in behaviour are not well understood in practice, as learning solvers are even more complex than traditional CP solvers. Thus, it is not yet clear to the research community under what circumstances learning is better, or even how to identify when or why a learning solver is performing poorly or not.

The aim of this paper is to cast some light on the practical behaviour of learning solvers by being able to better *profile their execution*. To achieve this, we instrumented the open-source LCG solver Chuffed [4] to provide statistical data regarding the clauses it learnt. We then fed this new data into the profiling tool introduced in [14], which we augmented with additional visualisations to display and analyse the LCG solving process. The long term aim of our research is to identify properties of the search that often indicate good or bad performance. If those properties can be identified, the profiler will be able to automatically focus the user's attention on the parts of the search that show those properties and suggest a reason for the behaviour, considerably simplifying the profiling task. As shown in Sect. 4, some of the information uncovered by the profiler has significant potential in this regard.

While using our augmented profiling tools on models where Chuffed achieved remarkably good performance, we realised that the clauses learnt by the solver could sometimes be used to modify the model itself, in such a way as to improve its execution for traditional CP solvers. This insight came from profiling clauses whose information was either (a) already expressed in the model by a single constraint, (b) not as strong as one would have expected, or (c) already captured by the model but not in an explicit way. Case (a) hints at a lack of appropriate propagation for a particular constraint in the model. The user might then decide to change the strength of the propagator (if the solver supports this) or modify the constraint to achieve better propagation. Case (b) also hints at a lack of propagation, possibly as part of the interaction between several constraints. The user might then decide to modify the constraints involved or add a new redundant constraint that achieves the desired propagation. Case (c) might suggest new information that could be expressed as a generic redundant constraint and which might increase propagation if added to the model (as it has increased propagation for the learning solver). While adding redundant constraints to a model is a well known method to improve performance, it can also have the opposite effect, depending on whether the redundant constraint helps propagation or not. Inferring useful, new redundant constraints for a given model is extremely difficult, and we are not aware of any proposed system or method capable of doing so. Using learnt clauses to achieve this is therefore an exciting new approach with significant potential. As shown in Sect. 3, we have already been able to detect clauses that fit in each of the three cases above, and modified the models accordingly obtaining considerable reductions in search effort.

2   Background

Constraint Programming: A finite domain *constraint problem* P is a tuple (C, D, f), where C is a set of constraints, D a *domain* which maps each variable

$x \in vars(C)$ to a finite set of integers $D(x)$, and f an objective function (if any). The set C is logically interpreted as the conjunction of its elements, while D is interpreted as $\wedge_{x \in vars(C)} x \in D(x)$. A literal of P is a unary constraint c where $var(c) \in vars(C)$. A CP solver starts from an original problem $P \equiv (C, D)$ and applies propagation to reduce domain D to D' as a fixpoint of all propagators for C. If D' is equivalent to *false*, we say P is failed. If D' fixes all variables, we have found a solution to P. Otherwise, the solver splits P into n subproblems $P_i \equiv (C \wedge c_i, D'), 1 \leq i \leq n$ where $C \wedge D' \Rightarrow (c_1 \vee c_2 \vee \ldots \vee c_n)$ and where c_i are literals (called *decision literals*), and iteratively searches these.

The search proceeds making decisions until either (1) a solution is found, (2) a failure is detected, or (3) a restart event occurs. In case (1) the search either terminates if the model has no objective function, or computes the value of the objective function f, sets a bound for the next value of f to be better (greater or smaller, depending on f) and continues the search for this better value. In case (2), the search usually backtracks to a previous point where a different decision can be made. In case (3) the search starts a new search tree, possibly incorporating new constraints learnt during the previous search.

Profiler: We use the functionality available in the profiler of [14], including its visualisation of the search tree and its tools for convenient navigation and analysis of the search. For example, Fig. 1 shows a search tree, where green diamonds denote solutions, red squares (and the highlighted yellow square) failures, grey squares nodes that are skipped due to backjumping, blue circles branching nodes, and white circles either unexplored nodes (in this case skipped due to a restart) or the root of an execution tree with restarts (as is the case in this execution). Labels showing the search decisions can be turned on or off for a given subtree or branch. Of particular interest is the capability to visually *merge* two search trees obtained by, for example, executing the same problem with
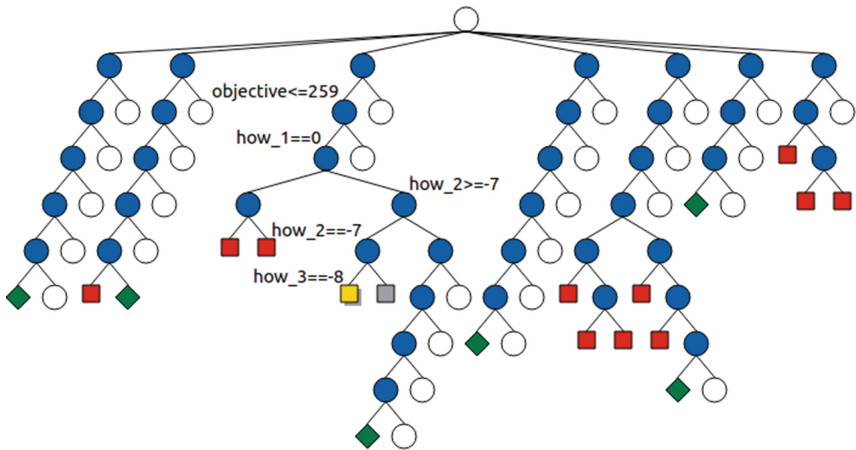
Fig. 1. Search tree for `freepizza` using Chuffed. The path to the highlighted node is labeled. (Color figure online)

two different solvers. The result is a combined tree, where the parts where the search is the same are visualised as usual, and those where the searches diverged are depicted as *pentagons* that can be expanded to show the divergent trees. This merging technique is particularly useful in combination with a *replaying* technique, where the search decisions used by a given solver when executing a problem are recorded, and the same decisions are then used to execute the same problem with a different solver. The merged tree then shows exactly where the two solvers behave differently in terms of constraint propagation.

Lazy Clause Generation: LCG solvers [5,10] can be seen as Satisfiability Modulo Theories solvers [9], where constraint propagators play the role of the theories. They extend CP solvers by instrumenting their propagators to explain the effect of propagation (i.e., domain changes) in terms of literals. In practice, these literals are all either *equality* ($x = d$ for $d \in D(x)$), *disequality* ($x \neq d$) or *inequality* ($x \geq d$ or $x \leq d$) literals. An *explanation* for literal ℓ is $S \rightarrow \ell$, where S is a set of literals. For example, the explanation for the propagator of constraint $x \neq y$ inferring literal $y \neq 5$ given literal $x = 5$ is $\{x = 5\} \rightarrow y \neq 5$. Explanations make the reasons behind constraint propagation explicit and can be used later when a failure occurs. In LCG solvers, each new literal inferred by a propagator is recorded in a stack in the order it was generated and attached to its explanation. Decision literals are also added to the stack and marked as such. This stack is called the *implication graph*. The *decision level* for any literal is the number of decision literals pushed in the stack before it. Thus, it is similar to the traditional concept of search tree depth in CP.

A nogood N is a set of literals that cannot be extended to a solution. Given an implication graph, LCG solvers compute a nogood by starting with the direct cause of the failure, and then eliminating literals by replacing them with their explanations until only one literal at the current decision level remains. The result is the 1UIP (First Unique Implication Point, [6]) nogood, and its negation ($\neg N$) is added as a clausal propagator. The search then backtracks to the decision level of the second latest literal in the nogood, where it applies the newly learnt clause. Importantly, if the second latest literal is not from the immediately preceding decision level, the search performs a *backjump*, skipping decisions that were unrelated to the failure.

Example 1. Consider the free pizza problem, where customers can get pizzas either by paying for them or by using vouchers. Each voucher is represented by a pair of numbers a/b, indicating the voucher allows customers to get b number of pizzas for free, as long as they pay for a number of pizzas. In addition, none of the b pizzas can be more expensive than the a pizzas. Given a customer who has m such vouchers and wants n pizzas, the aim is to minimise the total price paid for the n pizzas. The model used in the annual MiniZinc Challenge (denoted as `freepizza`) is as follows:

```
1  int: n;      set of int: PIZZA = 1..n;     % number of pizzas wanted
2  array[PIZZA]  of int: price;               % price of each pizza
3  int: m;      set of int: VOUCHER = 1..m;   % number of vouchers
4  array[VOUCHER] of int: buy;                % buy this many to use voucher
5  array[VOUCHER] of int: free;               % get this many free
6
7  set of int: ASSIGN = -m .. m; % i -i 0 (pizza free/paid with voucher i or not)
8  array[PIZZA] of var ASSIGN: how;
```

```
 9   array[VOUCHER] of var bool: used;
10
11   constraint forall(v in VOUCHER)
12   (used[v]<->sum(p in PIZZA)(how[p]=-v) >= buy[v]);
13   constraint forall(v in VOUCHER)(sum(p in PIZZA)(how[p]=-v) <= used[v]*buy[v]);
14   constraint forall(v in VOUCHER)(sum(p in PIZZA)(how[p]=v) <= used[v]*free[v]);
15   constraint forall(p1, p2 in PIZZA)((how[p1] < how[p2] /\ how[p1]= -how[p2])
16                                     -> price[p2] <= price[p1]);
17   int: total = sum(price);
18   var 0..total: objective = sum(p in PIZZA)((how[p] <= 0)*price[p]);
```

The first 5 lines introduce the parameters: lines 1 and 3 introduce n and m, respectively, line 2 introduces an array for the prices of the pizzas, while vouchers are introduced via two arrays in lines 4 and 5, where the ith voucher a/b is represented as buy[i]/free[i]. The next 3 lines define the variables: line 9 defines an array of vouchers, where used[v] is true iff voucher v was used. Line 8 defines an array of pizzas, where how[p] has value v if pizza p was free thanks to voucher v, has value 0 if p was paid for and not used in any voucher, and has value -v if p was paid for and used to get free pizzas with voucher v.
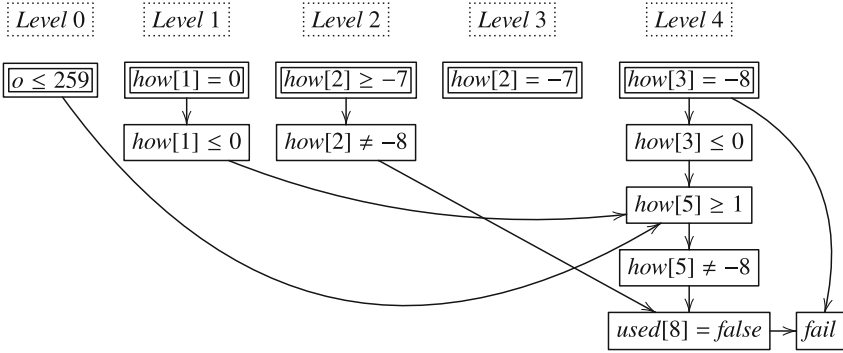
Fig. 2. Part of the implication graph for freepizza. Decision literals are double boxed.

Constraints start in line 11, which states that if voucher v was used (used[v] holds), then the total number of pizzas bought and assigned to v must be greater than or equal to the number of pizzas required by it (buy[v]). The constraint in line 12 states similar information but in the opposite direction: the total number of pizzas bought and assigned to voucher v must be less than or equal to used[v]*buy[v]. Together they constrain the total number of pizzas bought for v to be equal to buy[v], if used. The constraint in line 13 states that the total number of free pizzas obtained thanks to voucher v must be smaller than or equal to the number of free pizzas allowed by v if used (used[v]*free[v]). The last constraint is in line 14 and states that if there are two pizzas p1 and p2 assigned to the same voucher with p2 being free and p1 being paid for (given how[p1] < how[p2] and how[p1] = -how[p2]), then the price of p2 must be lower than or equal to that of p1. Finally, the objective function is defined as the sum of the prices of the pizzas that are bought. Figure 1 shows

a search tree for the execution of the model using Chuffed with the following input data:

```
n = 5;
price = [17, 98, 76, 36, 69];
m = 8;
buy = [4, 4, 1, 4, 2, 1, 1, 3];
free = [2, 4, 1, 1, 4, 2, 3, 3];
```

The third branch of the tree shows a restart after bound 259 has been established for the `objective`. Note that during the MiniZinc compilation process variable names are modified and, thus, variables `how_i` and `used_j` in the tree correspond to variables `how[i]` and `used[j]` in the model, respectively. Figure 2 shows part of the implication graph used to derive a nogood after the failure caused by search decision `how[3]=-8`. The failure set is {`how[3]=-8, used[8]=false`}. Since the nogood has two literals belonging to the current decision level (level 4), the last literal is reduced obtaining {$\mathtt{how}[3] = -8, \mathtt{how}[2] \neq -8, \mathtt{how}[5] \neq -8$}. Since this set still has two literals belonging to the current decision level, literal $\mathtt{how}[5] \neq -8$ is reduced to obtain {$\mathtt{how}[3] = -8, \mathtt{how}[2] \neq -8, \mathtt{how}[5] \geq 1$}. This reduction process continues until only one literal remains at the current decision level, yielding the 1UIP nogood {$\mathtt{objective} \leq 259, \mathtt{how}[1] \leq 0, \mathtt{how}[2] \neq -8, \mathtt{how}[3]=-8$}. This nogood is negated and added to the search as clause {$\mathtt{objective} > 259, \mathtt{how}[1] > 0$, $\mathtt{how}[2] = -8, \mathtt{how}[3] \neq -8$}, which is interpreted as the disjunction of its literals and prevents the same failure from recurring. After adding the clause, the search backjumps to level 2, as the nogood has no literal at level 3. This jump indicates that the decision at level 3 (`how[2]=-7`) is unrelated to the failure.

3   Exploring the Most Effective Learnt Clauses

The merging and replaying techniques mentioned above are useful when trying to understand the reasons for the success (or failure) of LCG solvers over traditional CP solvers. To remove the confounding effect of the different search orders, we can execute the LCG solver Chuffed first and replay its recorded decisions to execute the same model and search with Gecode [11], an efficient traditional CP solver. In general, for most constraints Gecode implements the same or stronger level of propagation as Chuffed. Therefore, the main differences when replaying the search in the form indicated above come from the clauses learnt by Chuffed. These clauses might help Chuffed (a) determine a failure earlier in the search, and/or (b) backjump further up than the parent node. In both cases, Gecode might perform extra search and, after merging the trees, those nodes will be displayed as pentagons by the tree visualisation.

We have modified Chuffed and the profiler to provide and visualise, respectively, extra information that is particularly useful when merging the replayed execution of an LCG solver by a CP solver. In particular, for each pentagon representing a failure node in the LCG execution, we can now compute and display the learnt clauses that helped to cause this failure. We refer to the number of pentagons to which a learnt clause contributed as *activity*, and measure its effectiveness in terms of search reduction, i.e., in terms of the number of nodes that

Gecode explores and Chuffed does not thanks to the addition of the clause. Since several clauses can contribute to a failure, our *reduced search* measure divides the total number of nodes by the number of clauses involved.

This information is collated at the end of the execution and shown to the user in table form (see Table 1 for an example). We used this method to explore Chuffed's behaviour on three different problems. As illustrated below, the information shown by our tables can lead to insights that result in effective model transformations.

3.1 First Case Study: `freepizza.mzn`

The first problem we explored combines the `freepizza` model from the MiniZinc Challenge 2015 as introduced in Example 1, with the following input data:

```
n = 10;
price = [70, 10, 60, 65, 30, 100, 75, 40, 45, 20];
m = 4;
buy = [1, 2, 3, 3];
free = [1, 1, 2, 1];
```

Table 1. Most effective learnt clauses in `freepizza`

Rank	Activity	Reduced search	Clause
1	159	3425	how[1] = -1 how[2] = -1 how[3] = -1 how[4] = -1 how[5] = -1
			how[1] = -2 how[2] = -2 how[3] = -2 how[4] = -2 how[5] = -2
			how[6] $\leq$ 0 how[6] $\geq$ 3
2	176	2068	how[7] $\leq$ 2 how[7] $\geq$ 4 how[1] $\neq$ -3 how[1] $\geq$ -2
3	34	1712	how[4] $\neq$ 3 how[1] = -3 how[2] = -3 how[3] = -3 how[4] = -3
4	8	1636	how[5] $\neq$ -3 how[3] $\neq$ 3
5	8	1636	how[8] $\neq$ -3 how[3] $\neq$ 3
6	8	1636	how[9] $\neq$ -3 how[3] $\neq$ 3
7	8	1636	how[10] $\neq$ -3 how[3] $\neq$ 3
8	143	1489	how[6] $\leq$ 2 how[6] $\geq$ 4 how[1] $\neq$ -3 how[1] $\geq$ -2
9	25	1404	how[5] $\neq$ -3 how[4] $\neq$ 3 how[4] $\leq$ 2
10	24	1403	how[10] $\neq$ -3 how[4] $\neq$ 3

We executed the problem using Chuffed, replayed its search using Gecode, merged their execution trees and explored the most effective learnt clauses in terms of reduced search and activity. Table 1 shows the top 10 clauses sorted by reduced search. We first concentrated on some of the shorter clauses, like `how[5]` $\neq -3$, `how[3]` $\neq 3$, which is ranked number four and states that pizza 3 cannot be obtained for free with voucher 3 by buying pizza 5 and assigning it to this voucher. This is a direct consequence of the constraint in line 14 and the fact that pizza 3 (cost 60) is more expensive than pizza 5 (cost 30). This helped us understand the longer clauses and realise that some of them were weaker (and more complex) than they should. Consider, for example, the top clause, which captures information about the relationship between obtaining pizza 6

with vouchers 1 or 2 (as $\mathtt{how}[6] \le 0, \mathtt{how}[6] \ge 3$ indicates that $\mathtt{how[6]}$ cannot be 1 or 2), and buying pizzas 1, 2, 3, 4, and 5, assigning them to these vouchers. It is clear by the input data that pizza 6 is more expensive than any other pizza and, thus, it cannot be obtained for free with any voucher (not just 1 and 2) and must be paid for. Therefore, the clause should be strengthened by expressing it as $\mathtt{how}[6] \le 0$. It was surprising to realise that this simple fact (and its cousin: the cheapest pizza cannot be used to obtain any other pizza for free) was not being learnt by the solver. This interesting insight reinforced the usefulness of studying the learnt clauses to better understand the information learnt (or not learnt). While the learnings ($\mathtt{how}[6] \le 0\ and\ \mathtt{how}[2] \ge 0$) were instance specific, the same ideas can be stated in a generic way and used as redundant constraints in the model:

```
% the most expensive pizza can never be bought with a voucher
constraint forall(p in PIZZA)
    (if forall(o in PIZZA where o != p)(price[p] > price[o])
        then how[p] <= 0 else true endif);
% the cheapest pizza can never be used with a voucher
constraint forall(p in PIZZA)
    (if forall(o in PIZZA where o != p) (price[p] < price[o])
        then how[p] >= 0 else true endif);
```

where ! = represents disequality. Of course, these redundant constraints will be vacuous if there is no single most expensive/cheapest pizza.

Another surprise was the fact that while many of the top clauses (4 to 7) were direct consequences of a single constraint (the one in line 14), learning them allowed Chuffed to avoid exploring significant amounts of nodes when compared to Gecode. We expected Gecode to also avoid exploring them by direct propagation. This indicated that the constraint was not propagating as strongly as expected. Upon inspection, it became clear that the how[p1] < how[p2] part of the constraint could be replaced by how[p2] > 0, indicating p2 is free. This is clearly stronger information and connects with the way the objective function is expressed, thus allowing stronger propagation when the objective is bounded. The modified constraint is:

```
constraint forall(p1,p2 in PIZZA)((how[p2]>0/\how[p1]= -how[p2])
                                -> price[p2] <= price[p1]);
```

To assess the model changes we randomly generated 100 input data files and measured the solving time using Gecode and Chuffed with fixed search (as specified in the original model) and with free search. Since we aimed to solve all instances to completion within a reasonable amount of time (not too easy, not too difficult) for both solvers, we generated the input data with between 2 to 10 vouchers for Gecode and 6 to 10 for Chuffed, each voucher requiring 1 to 4 pizzas to be paid for and allowing customers to get 1 to 4 pizzas for free. We also used 9 to 10 pizzas for Gecode with fixed search, 12 to 13 for Gecode with free search, and 15 to 16 for Chuffed. We excluded from the final results any problem data file that, for a given solver and search, was solved in under one second for all models. Thus, we used 74 and 80 data files for Gecode with fixed and free search, respectively, and 98 and 95 data files for Chuffed.

Table 2. Aggregate Results for Free Pizza over a set of random instances (relative)

		Models Ratio	GeoMean(time)	Median(time)	GeoMean(fails)	Median(fails)
Fixed Search	Chuffed	redundant/original	0.4885	0.5497	0.5186	0.5737
		final/redundant	0.7746	0.7905	0.9159	0.9368
		final/original	0.3784	0.4152	0.9368	0.5199
	Gecode	redundant/original	0.0904	0.1250	0.0925	0.1234
		final/redundant	0.0569	0.0786	0.0435	0.0461
		final/original	0.0051	0.0056	0.0040	0.0042
Free Search	Chuffed	redundant/original	0.7039	0.7162	0.7625	0.7426
		final/redundant	0.8228	0.8070	0.8872	0.8944
		final/original	0.5791	0.5830	0.6765	0.6876
	Gecode	redundant/original	0.1526	0.1468	0.1493	0.1459
		final/redundant	0.7205	0.7330	0.7991	0.8104
		final/original	0.1100	0.1050	0.1193	0.1187

Aggregated results for these data files are shown in Table 2, which compares the performance of the two solvers in terms of execution time and number of failures using three models: the original one, the one obtained by adding the two redundant constraints, and the final one obtained by also modifying the constraint in line 14 as indicated above. Clearly, our modifications improved the performance of both solvers (as all numbers are below 1), with the results being particularly significant for Gecode with fixed search, where the difference reaches two orders of magnitude. Detailed results are presented in Fig. 3, where each dot shows the solving time for a given data file using the original and the final models in the horizontal and vertical axes, respectively. The scatter plots show that the vast majority of the instances lie below the identity line and, thus, that our final model consistently performs better than the original one.

3.2   Second Case Study: `radiation.mzn`

The second problem we explored is the **intensity-modulated radiation therapy (IMRT) problem** [2], where radiation is given through repeated exposures of a device that delivers a rectangular field of radiation of uniform intensity. This rectangular field is shaped using horizontal lead rods positioned at the left and right of the rectangle, and which can slide laterally to block the radiation. In each exposure, the rods are moved into a given position, the radiation source switched on for a specified length of time and then switched off, to move to a new position. The model we studied is the one used in the MiniZinc Challenge 2015, where the input data is an $m \times n$ matrix `Intensity` of non-negative integers, where `Intensity[i, j]` represents the total amount of exposure that the cell in row i, column j should receive. The problem is to find a decomposition of the matrix into a positive linear combination of binary matrices, each with the
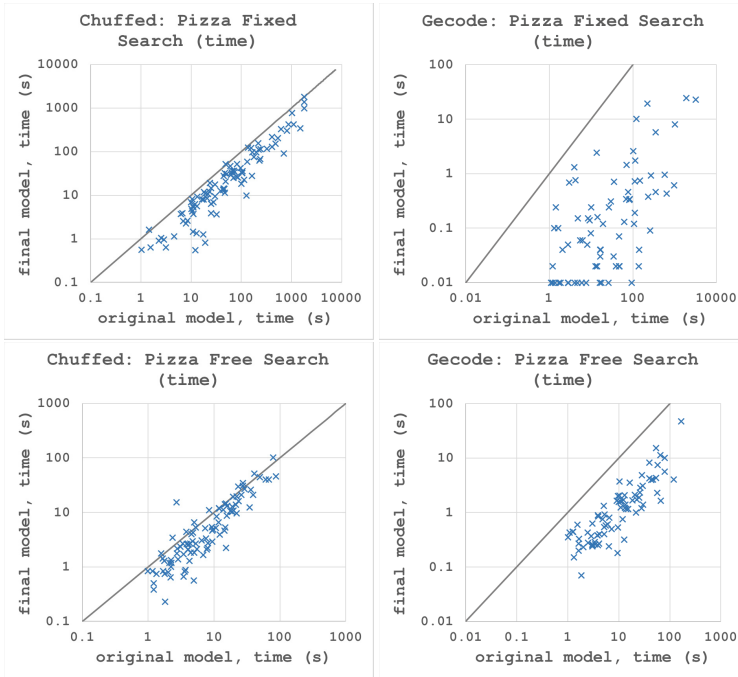
Fig. 3. Execution time of original and improved pizza models (logarithmic scale)

consecutive-ones property (i.e., all 1 s in any row are consecutive), where the 0 s represent the part of the row occluded by the rods and the 1 s the part that exposes radiation. The model is:

```
1   int: m;   % Rows
2   int: n;   % Columns
3   set of int: Rows    = 1..m;
4   set of int: Columns = 1..n;
5   array[Rows, Columns] of int: Intensity; % Intensity matrix
6   set of int: BTimes = 1..Bt_max;
7   int: Bt_max   = max(i in Rows, j in Columns) (Intensity[i,j]);
8   int: Ints_sum = sum(i in Rows, j in Columns) (Intensity[i,j]);
9
10  var 0..Ints_sum: Beamtime; % Total beam-on time
11  var 0..m*n: K; % Number of shape matrices
12      % N[b] is the number of shape matrices with associated beam-on time b
13  array[BTimes] of var 0..m*n: N;
14      % Q[i,j,b] is the number of shape matrices with associated beam-on time
15      % b that expose cell (i,j)
16  array[Rows, Columns, BTimes] of var 0..m*n: Q;
17
18  constraint
19      Beamtime = sum(b in BTimes) (b * N[b])
20      /\
21      K = sum(b in BTimes) (N[b])
22      /\
23      forall(i in Rows, j in Columns)
24          ( Intensity[i,j] = sum([b * Q[i,j,b] | b in BTimes]) )
25      /\
26      forall(i in Rows, b in BTimes)
27          ( upper_bound_on_increments(N[b], [Q[i,j,b] | j in Columns]) );
```

```
28
29   predicate upper_bound_on_increments(var int: N_b, array[int] of var int: L) =
30       N_b >= L[1] + sum([ max(L[j] - L[j-1], 0) | j in 2..n ]);
31
32   int: obj_min = lb((m*n + 1) * Beamtime + K);
33   int: obj_max = ub((m*n + 1) * Beamtime + K);
34   var obj_min..obj_max: objective = (m*n + 1) * Beamtime + K;
```

The first 7 lines introduce the parameters of the problem: lines 1 and 2 introduce m and n, respectively, line 5 introduces the intensity matrix, line 7 computes in Bt_max the maximum intensity value in the matrix, and in Ints_sum the sum of all intensity values in the matrix, which is an upper bound to the total amount of time the radiation beam will have to be on. The next lines introduce the variables of the problem: line 10 defines the total beamtime Bt_max for the solution, line 11 defines the total number K of binary matrices in the solution (which has $m \times n$ as upper bound), line 13 defines vector N, where variable N[b] is the number of matrices with the same beamtime b, and line 16 defines array Q, where variable Q[i,j,b] is the number of binary matrices with beamtime b that expose cell (i,j) to radiation.

Constraints start on line 19, which states that the total beamtime is the result of adding the beamtime used for every binary matrix. Line 21 states that the total number of matrices is the result of adding those used for every beamtime. Line 23 states that the intensity required by each cell (i,j) in the intensity matrix must be achieved by the solution, that is, it must be equal to the sum of beamtimes for each of the binary matrices that expose that cell. Finally, line 26 states the consecutive-ones property of the binary matrices by ensuring that for every beamtime b and every row i of Q[i,j,b], the number of times the intensity increases from a column j−1 to the next j, is equal or less than the number of binary matrices with that beamtime N[b].

Table 3 shows the 5 most effective learnt clauses obtained by executing the radiation model with the following input:

```
m = 9;  n = 9;  % rows and columns
Intensity = [|  4,  8, 11,  2,  5,  7,  1, 10,  4 |
               11,  4,  4,  5,  1,  8,  9,  3,  9 |
                2,  9,  6,  2,  4,  1,  5,  2,  6 |
               11,  9,  8,  9,  3,  2, 11,  6,  7 |
                2,  8, 11,  2, 10,  5,  5,  4,  5 |
                5,  9,  8,  1,  6,  3,  5, 11,  5 |
                ...
                7,  1,  6, 10,  0,  8,  1,  0,  0 |];
```

The top clause in Table 3 states that there should be a matrix that exposes cell [2,4] for a beamtime of 1, 3 or 5. This is because the input data requires the amount of radiation received by cell [2,4] to add up to exactly 5 units, which is an odd number. Thus, there needs to be at least one matrix with an odd beamtime. In particular, for 5 this requires a matrix with beamtime 1, 3, or 5, with anything longer than 5 resulting in the overexposure of the cell. This observation can be expressed in the model as follows:

Table 3. Most effective learnt clauses in `radiation`

Rank	Activity	Reduced search	Clause
1	3	378	$Q_{2,4,1} \geq 1$ $Q_{2,4,5} \geq 1$ $Q_{2,4,3} \geq 1$
2	3	378	$Q_{2,6,1} \geq 2$ $Q_{2,6,1} \leq 0$ $Q_{2,6,7} \geq 1$ $Q_{2,6,8} \geq 1$ $Q_{2,6,5} \geq 1$
			$Q_{2,6,2} \geq 4$ $Q_{2,6,4} \geq 2$ $Q_{2,6,3} \geq 1$
3	3	378	$Q_{2,1,1} \geq 1$ $Q_{2,1,5} \geq 1$ $Q_{2,1,3} \geq 1$ $Q_{2,1,9} \geq 1$ $Q_{2,1,10} \geq 1$
			$Q_{2,1,11} \geq 1$ $Q_{2,1,8} \geq 1$ $Q_{2,1,7} \geq 1$ $Q_{2,1,3} \geq 2$
4	2	315	$Q_{2,9,2} \leq 0$ $Q_{2,9,3} \geq 1$ $Q_{2,9,4} \geq 2$ $Q_{2,9,5} \geq 1$
			$Q_{2,9,6} \geq 1$ $Q_{2,9,7} \geq 1$ $Q_{2,9,8} \geq 1$ $Q_{2,9,9} \geq 1$
5	1	245	$(Q_{2,9,2} - Q_{2,8,2}) \geq 1$ $(Q_{2,9,1} - Q_{2,8,1}) \geq 2$
			$Q_{2,9,4} \geq 1$ $Q_{2,9,3} \geq 2$ $Q_{2,9,1} \geq 5$ $Q_{2,9,7} \geq 1$
			$Q_{2,9,8} \geq 1$ $Q_{2,9,9} \geq 1$ $Q_{2,9,6} \geq 1$ $Q_{2,9,5} \geq 1$
			$(Q_{2,9,1} - Q_{2,8,1}) \geq 2$ $(Q_{2,9,2} - Q_{2,8,2}) \geq 3$
			$(Q_{2,9,3} - Q_{2,8,3}) \geq 1$

```
constraint
  forall(b in BTimes where b mod 2 = 1)
    (forall(i in Rows, j in Columns where Intensity[i, j] = b)
      (sum([Q[i,j,k] | k in 1..b where k mod 2 = 1]) > 0));
```

While adding this constraint reduces the amount of search space explored, the reduction is small (3.0 % measured as median over 20 random instances), and is outweighed by the cost of propagating the extra constraints resulting in a 4.6 % longer execution. This suggests that Chuffed's good performance on this problem is not due to the learnt clauses, but to its conflict analysis (as confirmed in Sect. 4).

3.3   Third Case Study: Golomb Ruler

A Golomb ruler of size n is a set of n integer marks on an imaginary ruler, such that no two pairs of marks are the same distance apart. An optimal ruler is one with minimum length; i.e. the largest mark is to be minimised. The MiniZinc benchmarks set contains a model for finding such rulers[1], with two arrays of variables, one holding n integer variables (the `marks`) with domain $0..n^2$, and the other holding $\frac{n(n-1)}{2}$ integer variables (the `differences`) with domain $1..n^2$. This model is known to be difficult for learning solvers. Indeed, we find that Gecode is consistently faster than Chuffed on this model (see "Original Model" in Table 5). Nonetheless, Chuffed requires fewer failures to solve the problem and, thus, we decided to examine Chuffed's behaviour to see if we could learn something to help improve the model.

The top of Table 4 shows the 5 most effective clauses learnt by Chuffed while searching for $n = 10$. All these clauses are of the form `mark[i] >= n & mark[i+1] >= n+1 -> mark[i+2] >= n+3`, for some `i` and `n`. (Note

---

[1] https://github.com/MiniZinc/minizinc-benchmarks.

Table 4. Most effective learnt clauses for Golomb Ruler (before and after the first modification)

	Rank	Activity	Reduced Search	Clause
Original	1	49	193	mark[6] ≥ 38, mark[5] ≤ 35, mark[4] ≤ 34
	2	6	170	mark[5] ≥ 18, mark[4] ≤ 15, mark[3] ≤ 14
	3	6	170	mark[5] ≥ 15, mark[4] ≤ 12, mark[3] ≤ 11
	4	5	168	mark[5] ≥ 22, mark[4] ≤ 19, mark[3] ≤ 18
	5	50	163	mark[6] ≥ 36, mark[5] ≤ 33, mark[4] ≤ 32
Modified	1	2	55	mark[6] ≥ 19, mark[4] ≤ 14, mark[3] ≤ 13
	2	3	55	mark[6] ≥ 18, mark[4] ≤ 13, mark[3] ≤ 12
	3	3	41	(mark[8] - mark[6] ≤ 5), (mark[8] - mark[6] ≥ 7), (mark[10] ≥ 55), (mark[6] ≤ 41), (mark[10] - mark[8] ≤ 5)

that a clause of the form $\{A, B, C\}$ can be interpreted as $\neg B \wedge \neg C \rightarrow A$.)
This indicates a missing constraint which the solver is effectively rediscovering.
Looking again at the problem, we confirmed it was correct to add the following redundant constraint: `mark[i] + 3 <= mark[i+2]`, for all i. Clearly
`mark[i+2]` is at least one more than `mark[i+1]`, which is at least one more
than `mark[i]`. Thus, `mark[i+2]` and `mark[i]` are at least two apart and, if
so, both intermediate differences must be one, which is forbidden.

We added this redundant constraint to the model, re-executed the modified model and examined the 5 most effective learnt clauses (see bottom of
Table 4). The first two follow the pattern `mark[i] >= n & mark[i+1] >=`
`n+1 -> mark[i+3] >= n+5`. The third illustrates a property of connected
differences: if the difference between `mark[i]` and `mark[j]` is e.g. 6, and the
difference between `mark[j]` and `mark[k]` is at least 6, then the difference
between `mark[i]` and `mark[k]` is at least 6+6+1=13. (The extra one appears
for the same reason as above.) All these clauses refer to the idea that the distance between two marks that are m positions apart is equal to the sum of the
inner distances, which are all different. As a result, this distance is at least as
large as the sum of the arithmetic sequence of natural numbers, i.e., $\frac{m(m+1)}{2}$. In
fact, by following this methodology we rediscovered a redundant constraint for
this problem that was earlier discussed in [3].

The inclusion of this redundant constraint reduces the search effort required
to solve the problem, both in number of nodes and in time (see "Improved
Model" in Table 5). Interestingly, even the non-learning solver Gecode benefited
from the constraint. This demonstrates how, even when learning solvers are not
the strongest for a given problem, we can gain useful insights from examining
their behaviour.

4 Profiling Statistics

In addition to computing and showing the most effective clauses in table form, we
have modified the profiler to compute statistical data based on the information

Table 5. Golomb Ruler Results

	Size n	Original Model		Improved Model	
		Time (s)	Number of Failures	Time (s)	Number of Failures
Chuffed	$n = 10$	1.99	20,912	1.49	19,343
	$n = 11$	72.36	307,957	54.25	288,071
	$n = 12$	616.2	2,329,959	512.63	2,254,206
Gecode	$n = 10$	0.74	23,463	0.57	19,928
	$n = 11$	15.81	374,886	12.09	321,419
	$n = 12$	147.00	3,002,474	117.83	2,656,663

provided by the solvers. In particular, for a learnt clause $L \equiv \{l_1, \ldots, l_m\}$ the profiler can now display the following information:

– **Length:** the number m of literals in L.
– **Decision level:** the decision level at which the failure occurred.
– **Total number of variables:** the cardinality of the set $vars(L)$. Note that this is always less than or equal to the length. The number of variables can be much smaller than the length if variables appear in many literals.
– **Literal Block Distance:** number of decision levels where literals in L were inferred. Note that this can never be larger than the decision level for L. This measure has been shown to be useful in SAT problems [1].
– **Backjump destination:** the decision level of the node to which the search backjumped after learning L.
– **Backjump distance:** the distance between the decision level of the node in which L was learnt and that of the backjump destination. Note that if it is 1, the behaviour is similar to traditional CP backtracking.
– **Activity:** number of times L is involved in inferring failures later in the search.
– **Size reduction:** number of nodes avoided thanks to L being learnt (both in terms of the tree that would have been explored by a CP solver and in terms of backjumped ones). This measure requires a comparison with Gecode's execution.
– **Generation time:** point in time during the search at which L was learnt.

Example 2. The length of the clause $\{$objective $> 259,$ how$[1] > 0,$ how$[2] = -8,$ how$[3] \neq -8\}$ found in Example 1 is 4, which is equal to its number of variables. Its decision level is 4, its literal block distance is also 4, its backjump destination is level 2, and its backjump distance is also 2.

As mentioned before, our aim is to determine whether any of this information could be used to explain the reasons behind good or bad performance and, thus, should be highlighted to users as possible markers for such behaviour. Figure 4 provides an example of the plots that display some (due to space limitations) of this statistical information. Each dot in each square represents a single clause. There is a scatter plot for each pair of attributes, arranged in a triangular matrix. The attributes shown are (in order from left to right and top to bottom) time at which the associated clause was learnt (in microseconds), decision level, backjump distance, literal block distance, backjump destination and raw activity.
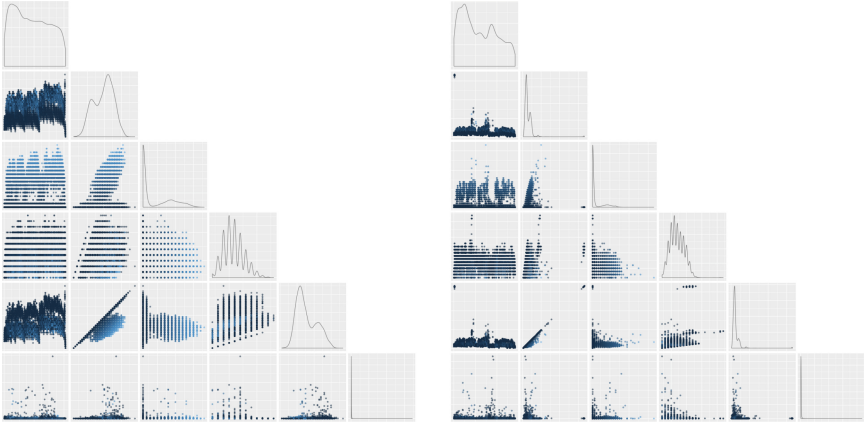
Fig. 4. Profiling plots for the `radiation` problem with Chuffed using fixed and free search.
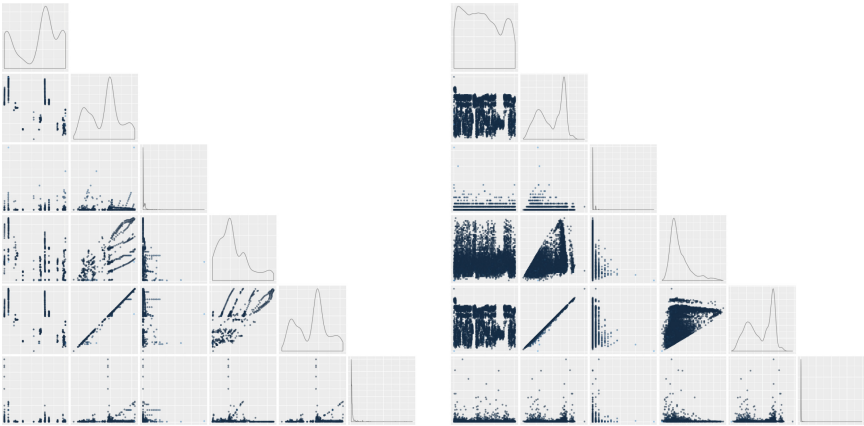
Fig. 5. Profiling plots for `cvrp` (left) and `opd` (right).

The plots along the diagonal show kernel density estimates of each attribute. For example, the plot shown in the fifth row and second column (denoted (5,2)) shows the backjump destination against the decision level, while plot (3,3) shows the overall distribution of backjump distances.

Let us consider Fig. 4, which shows the statistics obtained for an instance of the `radiation` problem model with fixed search (left) and free search (right), with Chuffed performing much better for the fixed search. Let us compare this with Fig. 5, which shows the statistics for two instances of the MiniZinc Challenge problems where Chuffed does not performs well: `cvrp` (left) and `opd` (right). There are a few interesting things to note. First, comparing the (6,1) plots, there is a relatively high level of activity throughout the entire execution of

`radiation`, while the level of activity in the other two problems is smaller and, in `cvrp`, only appears at particular points in time.

From the plots we can also observe the high-level behaviour of the search. For `cvrp` and `opd` the (5,2) plots shows quite a compact diagonal, indicating that the backjump destination level of most clauses is relatively similar to their decision level – in other words, the search does not backjump significantly. This is not the case for `radiation`, where there is a significant "bulge" below the diagonal.

The `radiation` problem on the left exhibits a related phenomenon, visible in plots (2,2) and (5,5), which show the distribution of decision level and backjump destination. In the fixed search, the peaks of these plots are the reverse of one another, a phenomenon unique among all the instances shown. This indicates that the search backjumps from a deeper level (the peak on the right) to a shallower level (the peak on the left). Indeed, the search is designed to encourage this behaviour; it fixes the variables in such a way that after the early "master" variables have been selected, the problem consists of wholly independent subproblems. Whenever one such subproblem fails, *all* subproblems can be discarded and the search returns to the master variables. This backjumping caused by conflict analysis is the reason that Chuffed performs well on this problem, as suggested in Sect. 3.2. The free search performs worse because this behaviour occurs less frequently. The plots of these statistics confirm that the specified fixed search is performing as expected.

Finally, Fig. 6 shows the profiling plot for the Golomb ruler problem examined in Sect. 3. We observe in the (3,3) plot that there is no significant backjumping during the search. This partially explains why Chuffed is slower than Gecode, and confirms that any benefit for the learning solver comes from the learnt clauses and not from any improvement to the search behaviour.

From these and other examples we have already identified some statistical markers of learning behaviour, including:

Fig. 6. Golomb ruler original model, $n = 11$

- Literal block distance being always close to the decision level: this indicates that failure explanations are poor and backjumping is likely to be minimal;
- Failure decision level being confined to a narrow range, especially deep in the tree: this is a clear indication of search making no progress;
- Deep decision levels coupled with low backjump distance and low learnt clause activity: this strongly suggests poor performance.

5   Conclusions

Learning solvers dramatically outperform traditional CP solvers on many problems, but their behaviour in practice is opaque and hard to understand. We have instrumented the learning solver Chuffed to give detailed information about its behaviour, so that it can be better understood. In particular, we have considered several case studies and shown how profiling leads to a better understanding of the solver's behaviour on each problem, and how the profiling information can lead directly to improvements of the model by either modifying its constraints or adding new redundant ones. One may argue that redundant constraints such as those derived in Sect. 3 could just as well be found without any profiling data. However, the method we show here allows the solver to tell us precisely the constraints it requires to reach its conclusion, avoiding the "guess and test" approach to model improvement.

This work prompts further analysis of learning behaviour. In particular, there is the possibility to include other features of learnt clauses, and to determine via machine learning techniques specific markers for good or bad performance. As well as providing feedback to the user, such indicators could be used to guide heuristics for solvers when performing autonomous search. It would also be interesting to apply the presented techniques to SMT solvers [9]. Further, while here we have focused on the statistical analysis of learned clauses, studying the clause graph structure, as for example in [8], can be insightful as well. The profiler and the modified versions of Chuffed and Gecode used in this work are available at http://www.minizinc.org.

References

1. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st International Joint Conference on Artifical Intelligence IJCAI 2009, pp. 399–404 (2009)
2. Baatar, D., Boland, N., Brand, S., Stuckey, P.J.: CP and IP approaches to cancer radiotherapy delivery optimization. Constraints **16**(2), 173–194 (2011)
3. Barták, R.: Effective modeling with constraints. In: Seipel, D., Hanus, M., Geske, U., Bartenstein, O. (eds.) INAP/WLP 2004. LNCS (LNAI), vol. 3392, pp. 149–165. Springer, Heidelberg (2005)
4. Chu, G.G.: Improving combinatorial optimization. Ph.d. thesis, The University of Melbourne (2011)
5. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 352–366. Springer, Heidelberg (2009)
6. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design ICCAD 1996, pp. 220–227. IEEE Computer Society, Washington, DC (1996)
7. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, pp. 530–535. ACM (2001)

8. Newsham, Z., Lindsay, W., Liang, J.H., Czarnecki, K., Fischmeister, S., Ganesh, V.: SATGraf: visualizing community structure in boolean SAT instances. In: Heule, M., Weaver, S. (eds.) Theory and Applications of Satisfiability Testing – SAT 2015. LNCS, vol. 9340, pp. 62–70. Springer, Heidelberg (2015)

9. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Abstract DPLL and abstract DPLL modulo theories. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 36–50. Springer, Heidelberg (2005)

10. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation = lazy clause generation. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 544–558. Springer, Heidelberg (2007)

11. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and programming with Gecode (2016). http://www.gecode.org

12. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Why cumulative decomposition is not as bad as it sounds. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 746–761. Springer, Heidelberg (2009)

13. Schutt, A., Stuckey, P.J., Verden, A.R.: Optimal carpet cutting. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 69–84. Springer, Heidelberg (2011)

14. Shishmarev, M., Mears, C., Tack, G., Garcia de la Banda, M.: Visual search tree profiling. Constraints **21**(1), 77–94 (2016)

15. Stuckey, P.J., Feydy, T., Schutt, A., Tack, G., Fischer, J.: The MiniZinc challenge 2008–2013. AI Mag. **35**(2), 55–60 (2014)

On Incremental Core-Guided MaxSAT Solving

Xujie Si[1(✉)], Xin Zhang[1], Vasco Manquinho[2], Mikoláš Janota[3],
Alexey Ignatiev[4,5], and Mayur Naik[1]

[1] Georgia Institute of Technology, Atlanta, USA
`six@gatech.edu`
[2] INESC-ID, IST, Universidade de Lisboa, Lisbon, Portugal
[3] Microsoft Research, Cambridge, UK
[4] LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Lisbon, Portugal
[5] ISDCT SB RAS, Irkutsk, Russia

Abstract. This paper aims to improve the efficiency of unsat core-guided MaxSAT solving on a sequence of similar problem instances. In particular, we consider the case when the sequence is constructed by adding new hard or soft clauses. Our approach is akin to the well-known idea of incremental SAT solving. However, we show that there are important differences between incremental SAT and incremental MaxSAT, where a straightforward implementation may lead to a sharp decrease in performance. We present alternatives that enable to cope with such issues. The presented algorithm is implemented and evaluated on practical problems. It solves more instances and yields an average speedup of 1.8× on previously solvable instances.

1 Introduction

MaxSAT is an optimization variant of the Boolean Satisfiability (SAT) problem. Recent years have witnessed vast improvements in the performance of MaxSAT solvers [1, 4–6, 14, 15, 24–26]. Emerging applications in a variety of domains pose large MaxSAT instances comprising tens of millions of clauses to such solvers.

A special but common scenario concerns applications which pose a *sequence* of *similar* large MaxSAT instances. For example, many applications involve a sequence of small updates to a large instance (e.g., verification via abstraction refinement [13, 28] or user interaction [18]). Alternatively, MaxSAT-based solvers pose such sequences in order to scale to ever larger instances (e.g., using lazy [16] or demand-driven [29] methods) or more expressive theories (e.g., MaxSMT [7] and Markov Logic Networks [17, 27]). Instead of solving each instance in the sequence from scratch, it is desirable to improve the efficiency of MaxSAT solvers by reusing results computed across invocations on such instances.

In this paper, we focus on an especially common case in which the sequence of MaxSAT instances is constructed by *adding* hard or soft clauses. Moreover, the new clauses are determined by the solution to the previous instance. We target an unsat core-guided algorithm [12] which forms the basis of many popular MaxSAT solvers. This algorithm solves a single MaxSAT instance by solving a sequence

of SAT instances until the underlying SAT solver finds a satisfying solution. Each SAT instance is constructed using the unsat cores discovered in previous SAT instances. Since adding clauses to the current MaxSAT instance does not invalidate existing unsat cores, a compelling idea to improve the performance of solving the resulting MaxSAT instance is to reuse the existing unsat cores.

Surprisingly, however, we observe that a naive implementation of this idea can fail to yield performance benefits or, even worse, sharply curtain them. This reflects an inherent challenge to making core-guided MaxSAT solving incremental[1]: for a MaxSAT instance formed with two disjoint sets of clauses ϕ and δ, solving ϕ followed by $\phi \cup \delta$, rather than solving $\phi \cup \delta$ directly, restricts the set of possible computations. This is because the set of unsat cores of ϕ is always a subset of those of $\phi \cup \delta$. Reusing the unsat cores of ϕ in solving $\phi \cup \delta$ can be detrimental because the MaxSAT algorithm's performance crucially depends on the quality of the unsat cores, and the unsat cores learnt from solving ϕ may be of poorer quality than those it would learn from solving $\phi \cup \delta$ directly.

To address this challenge, we propose a hybrid solving framework that alternates between the incremental algorithm and its non-incremental version. In each iteration, our framework checks whether the current instance may potentially benefit from reusing the cores learnt on previous instances. If the check succeeds, it applies the incremental algorithm by reusing such cores. Otherwise, it discards the cores learnt thus far and applies the non-incremental algorithm.

We implemented our approach in the Open-WBO MaxSAT solver [22] and evaluated it on 74 sequences generated from diverse applications in verification and information retrieval. Together, these sequences contain 669 MaxSAT instances, with an average of 10 million clauses per instance. Our evaluation shows that our approach outperforms the baseline approaches significantly: it yields an average speedup of $1.8\times$ per sequence over the non-incremental approach, and it solves 19 more sequences than the naively-incremental approach.

2   Preliminaries

A propositional formula in *Conjunctive Normal Form (CNF)* is a conjunction of *clauses* where each clause is a disjunction of *literals*. A literal is either a Boolean variable x_i or its negation $\neg x_i$. A literal x_i ($\neg x_i$) is valued to true if x_i is assigned to true (false). A literal x_i ($\neg x_i$) is valued to false if x_i is assigned to false (true). A clause is said to be *satisfied* if at least one of its literals is valued to true. If all literals in a clause are valued to false, the clause is said to be *unsatisfied*. We refer to CNF formulas as sets of clauses and clauses as sets of literals. For a CNF ϕ, the *Satisfiability (SAT)* problem is defined as finding an assignment to all variables in ϕ that satisfies all clauses or determining that such an assignment does not exist.

---

[1] Some works (e.g., [20]) define "incremental MaxSAT solving" as solving a MaxSAT instance by using a SAT solver incrementally. In this paper, it denotes solving a MaxSAT instance by reusing the results of solving another similar MaxSAT instance.

The *Maximum Satisfiability (MaxSAT)* problem is an optimization version of SAT. Given a CNF formula ϕ, the goal is to find a total assignment that minimizes the number of unsatisfied clauses. In *partial MaxSAT*, the CNF formula $\phi = \phi_S \cup \phi_H$ contains a set of *soft clauses* ϕ_S and a set of *hard clauses* ϕ_H. The goal is to find an assignment such that all hard clauses are satisfied while minimizing the number of unsatisfied soft clauses. Finally, a *weighted clause* is a pair (c, w) where $w \in \mathbb{N}$ is the cost of not satisfying the clause c. In *weighted partial MaxSAT*, the goal is to find a total assignment where all hard clauses are satisfied, while minimizing the sum of the weights of unsatisfied soft clauses. In the remainder of the paper, we use MaxSAT to refer to the more general problem of weighted partial MaxSAT.

Most of the state-of-the-art MaxSAT algorithms rely on successive calls to a SAT solver. In particular, *Core-Guided* MaxSAT algorithms have been shown to be very effective in solving instances that arise from real-world applications [23]. These algorithms take advantage of the ability of SAT solvers to identify unsatisfiable subformulas (also known as *unsatisfiable cores*).

A SAT solver call $\mathtt{SAT}(\phi, \mathcal{A})$ receives a CNF formula ϕ and a set of assumptions $\mathcal{A}$. The set $\mathcal{A}$ defines a set of literals that must be true in the model of ϕ returned by the SAT call. A SAT call returns a triple $(\mathsf{st}, \nu, \phi_C)$ where st denotes the solver status (SAT or UNSAT). If the call is satisfiable, then ν contains a model of ϕ. Otherwise, $\phi_C \subseteq \phi$ contains a *core*: an unsatisfiable subformula of ϕ. Note that a SAT call can return UNSAT, even when ϕ is satisfiable. This occurs when there is no model of ϕ such that all assumption literals in $\mathcal{A}$ can be set to true. In this case, ϕ_C contains clauses from ϕ as well as literals from $\mathcal{A}$.

3   Sequential Maximum Satisfiability

We define the *sequential MaxSAT problem* as the problem of solving a sequence of n MaxSAT formulas $\phi^1, \phi^2, \ldots, \phi^n$, with $\phi^k \subseteq \phi^{k+1}$. This problem arises in many applications [7,16,18,28], where a sequence of MaxSAT instances are to be solved. In most cases, the k-th MaxSAT instance ϕ^k is generated by incrementally modifying the previous instance ϕ^{k-1}, based on the solution of ϕ^{k-1}.

A straightforward solution to the sequential MaxSAT problem is to use any off-the-shelf MaxSAT solver to *independently* solve each MaxSAT instance ϕ^k ($1 \leq k \leq n$). This is the approach currently used in most applications [7,16,18,28]. However, this does not enable reusing information obtained from solving a given formula in solving the subsequent formulas.

This section is organized as follows. Section 3.1 reviews the *Fu & Malik* algorithm for MaxSAT with incremental SAT—previously published in [20]. Sections 3.2 and 3.3 form the core contribution of the paper: Sect. 3.2 shows how to generalize *Fu & Malik* to solve sequential MaxSAT and Sect. 3.3 introduces restarts to cope with performance issues in the introduced algorithm.

Algorithm 1. Fu-Malik Algorithm with Incremental SAT [20]

Input: $\phi = \phi_H \cup \phi_S$

Output: optimal solution to ϕ

```
1  φ_W ← φ_H ∪ {c ∪ {blockingVar(c)} | c ∈ φ_S}        // fresh blocking variables
2  A ← {¬blockingVar(c) | c ∈ φ_S}                     // enable all soft clauses
3  while true do
4  │   (st, ν, φ_C) ← SAT(φ_W, A)
5  │   if st = SAT then return ν                        // optimal solution to φ
6  │   V_R ← ∅
7  │   m_C = min{weight(c) | c ∈ φ_C ∧ soft(c)}
8  │   foreach c ∈ φ_C ∧ soft(c) do
9  │   │   V_R ← V_R ∪ {r}                              // r is a fresh relaxation variable
10 │   │   c_r ← (c \ {blockingVar(c)}) ∪ {r} ∪ {b_r}   // b_r is a fresh variable
11 │   │   A ← A ∪ {¬b_r}                               // enable c_r
12 │   │   φ_W ← φ_W ∪ {c_r}
13 │   │   weight(c_r) ← m_C
14 │   │   if weight(c) > m_C then weight(c) ← weight(c) − m_C
15 │   │   else  A ← (A \ {¬blockingVar(c)}) ∪ {blockingVar(c)}  // disable c
16 │   φ_W ← φ_W ∪ {CNF(∑_{r∈V_R} r ≤ 1)}
```

3.1 Background: *Fu & Malik* MaxSAT Algorithm

The *Fu & Malik* algorithm [12] was initially proposed in 2006 and later extended to weighted MaxSAT [3,19]. More recently, a new version was proposed where the SAT solver is not rebuilt in each iteration, thus allowing the reuse of knowledge learnt by the SAT solver in previous iterations. Hence, the SAT solver is used incrementally for a *single* MaxSAT instance. Later we will extend this to use the whole MaxSAT solver incrementally, i.e. for multiple MaxSAT instances.

Algorithm 1 reviews the pseudo-code of *Fu & Malik* for solving weighted partial MaxSAT using SAT incrementally [20]. The working formula ϕ_W is initialized to all hard clauses with all soft clauses extended with a fresh *blocking variable*. Negations of the blocking variables are added to the assumptions $\mathcal{A}$, thus *enabling* the original soft clauses (lines 1–2). When a soft clause c is extended with a blocking variable b to form $(c \vee b)$, then adding $\neg b$ to the assumptions effectively enables c since the SAT solver must necessarily satisfy c. Conversely, adding b to the assumptions *disables* c since $(c \vee b)$ is trivially satisfied.

Each iteration issues a SAT call on line 4. If the working formula is satisfiable, the optimal solution was found. Otherwise, ϕ_C is an unsatisfiable subformula (core). In this case, for each soft clause c in ϕ_C, a new relaxed clause c_r is created from c with two additional variables (a relaxation and a blocking variable). If the clause is enabled through the blocking variable, then the relaxation variable represents if the original clause is satisfied (or not) in the MaxSAT solution.

On line 7, the *weight of the core* m_C is the minimum weight of all soft clauses in ϕ_C. Soft clauses $c \in \phi_C$ with weight equal to m_C are disabled (line 15) and

replaced with their relaxation c_r. Soft clauses $c \in \phi_C$ with weight larger than m_C are not removed. Their weight is decreased by m_C, thus resulting in a *clause split*, since the original weight is divided between c and its relaxation c_r.

Finally, note that since the working formula is always expanded, the SAT solver is never rebuilt and its internal state is kept (including the learnt clauses).

3.2 Our Approach: Solving Sequential MaxSAT Incrementally

In this section we propose how to solve a sequential MaxSAT problem incrementally. Consider a sequence of MaxSAT formulas $\phi^1, \phi^2, \ldots, \phi^n$, with $\phi^k \subseteq \phi^{k+1}$.

We apply Algorithm 1 to ϕ^1 and then extend the resulting working formula ϕ_W with hard clauses from $\phi_H^2 \setminus \phi_H^1$, and, soft clauses from $\phi_S^2 \setminus \phi_S^1$ each extended with a fresh blocking variable. Then resume the main loop of Algorithm 1 (from line 3). This process is analogously repeated for the upcoming formulas in the sequence. More precisely, each time ϕ_W becomes satisfiable, the clauses $\phi_H^{k+1} \setminus \phi_H^k$, and $\phi_S^{k+1} \setminus \phi_S^k$ are added to ϕ_W, where the soft clauses are adorned with a fresh blocking variable, which is in turn reflected in the assumptions. Then go to line 3.

As such, it is not necessary to restart the search from scratch for each formula in the sequence. This approach is correct because the addition of new soft or hard clauses does not invalidate any of the previously found cores. Note that the approach is incremental at two levels: it uses the SAT solver incrementally for each instance but also is incremental across the sequence of instances.

3.3 Extending Sequential MaxSAT Solving with Restarts

Consider a sequence of MaxSAT instances $\phi^1, \ldots, \phi^n$ where $\phi^i \subseteq \phi^j$ for $1 \leq i < j \leq n$. When solving ϕ^i first, the incremental *Fu & Malik* has a "narrower perspective" than the non-incremental *Fu & Malik* applied directly on ϕ^j. More specifically, the set of possible cores in ϕ^i is always a subset of the possible cores in ϕ^j. Consequently, the incremental version may end up finding a core of *poorer quality* than the non-incremental version. Finding a core of poor quality is often detrimental to the rest of the computation. This is especially true for the weighted *Fu & Malik*, which splits clauses based on the minimum weight of the found core. This is illustrated by Example 1.

Example 1. Consider an $n \in \mathbb{N}$, weights $w_1 < w_2 \in \mathbb{N}$, and the core $\mathcal{C}[w_1, w_2] = \{(w_2, \neg a_i) \mid i \in 1..n\} \cup \{(w_1, b \vee \bigvee_{i \in 1..n} a_i), (w_2, \neg b)\}$. Once found, this core is conceptually split into the sets of clauses $\mathcal{C}[w_1, w_1]$ and $\mathcal{C}[0, w_2 - w_1]$, where the first set is relaxed. This creates $n + 1$ new clauses and relaxation variables, incurring thus cost on further computation. If the next iteration adds the hard clause (b), then the MaxSAT solver can use the simpler core $\{(b), (w_2, \neg b)\}$ without encountering the large core above.

Here we propose a solution to the above-outlined issue, which is to *restart* the whole computation once we suspect that the incremental version is finding

cores of poor quality. We say that a given soft clause $c \in \phi_C$ is *split* if its weight is larger than the weight of the unsatisfiable core (m_C). If a clause c is split, it means that an unsatisfiable core with other soft clauses with smaller weights was found. In our solver, we maintain a *split counter* for every soft clause in the formula and define a split limit. When the split limit is reached for some soft clause, the solver is rebuilt and Algorithm 1 is restarted. In order to maintain completeness, the solver restarts at most once for each MaxSAT formula ϕ^i in the sequential MaxSAT instance.

4   Empirical Evaluation

We evaluate our technique on sequential MaxSAT problems generated from three applications: abstraction refinement, user-guided analysis, and statistical relational inference. Abstraction refinement [28] tackles a central problem in software verification: finding a program abstraction that only tracks information relevant to proving assertions of interest. It solves a sequence of MaxSAT instances to construct such an abstraction. User-guided analysis [18] iteratively incorporates user feedback in software analysis tools to eliminate false alarms. In each iteration, it solves a MaxSAT instance to infer the most likely set of true alarms based on the current feedback. Statistical relational inference [16] enables a wide range of information retrieval tasks by solving a system of weighted first-order constraints over a relational database. It scales to large database instances by lazily solving a sequence of progressively growing MaxSAT instances.

We implemented our technique in the Open-WBO [22] MaxSAT solver. All experiments were done on a Linux machine with a 3.0 GHz processor. We limited each MaxSAT solver invocation to 32 GB RAM and 30 min of CPU time.

We compare our incremental algorithm with restarts to two baselines: the non-incremental and the incremental-without-restarts algorithms. The former is the original Open-WBO solver while the latter is obtained by disabling restarts in our solver. To evaluate the effect of different split limits, we use the split limits 2, 5, 10, and 15. We generated the sequential MaxSAT instances by running the applications with both our solver (using split limit of 5) and the non-incremental solver until the application terminates or any MaxSAT invocation exceeds one hour. Since the solutions returned by the MaxSAT solver may affect the sequence of MaxSAT instances generated by the applications, we used both the solvers to reduce the bias introduced by a particular solver in the instance generation. Following this recipe, we obtained 74 sequential MaxSAT problems comprising 669 MaxSAT instances. The number of clauses in each MaxSAT instance ranges from two thousand to 150 million, with 10 million being the average.

The cactus plot in Fig. 1(a) shows the number of sequential MaxSAT instances solved by our approaches and the baseline approaches within given CPU times. As the plot shows, our incremental algorithm with 5 as the split limit solves the most instances.

Moreover, on the instances that can be solved by both approaches, our approach with 5 as the split limit yields an average speedup of $1.8\times$ over the non-incremental approach. On certain instances, the benefit is as high as $4.7\times$.
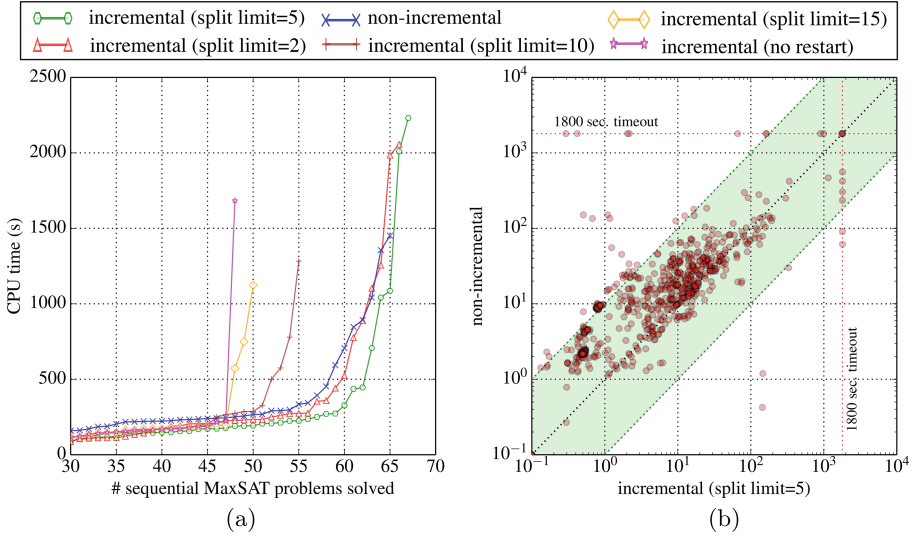
Fig. 1. Performance of our approaches and baseline approaches on (a) sequential MaxSAT problems and (b) each individual MaxSAT instance in the sequences.

The scatter plot in Fig. 1(b) further compares the time consumed by both approaches on each individual MaxSAT instance. As the plot shows, the speedup can be as high as $296\times$ on certain instances. This shows that our approach effectively improves the overall performance by reusing computation across similar MaxSAT instances in the same sequence.

We also observe that the incremental algorithm without restarts performs significantly worse compared to other approaches. This justifies the need for restarts in incremental MaxSAT solving: naively reusing cores computed from previous smaller instances can severely impede the solver's performance on the current instance. On the other hand, our approach effectively avoids this problem by restarting the solving process when it observes any clause being split too often.

We further observe that using a split limit that is too high (e.g., 10 or 15) or too low (e.g., 2) adversely affects the performance of the incremental algorithm. When the cores learnt from previous smaller MaxSAT instances are unsuitable for the current MaxSAT instance, a too high split limit can either fail to trigger the restart or only triggers the restart after the algorithm has spent significant time running with these cores. On the other hand, using a too low limit can trigger the restart too often, making the algorithm fall back to its non-incremental version. While finding an adequate restart condition is an interesting research direction, using 5 as the split limit yields the best overall performance on the evaluated instances.

5 Discussion and Future Work

Incrementality and restarts are well established in SAT solving [10,11], so a natural question is why they do *not* directly translate to MaxSAT. Adding new clauses to a SAT solver does not invalidate existing learnt clauses just as new clauses do not invalidate existing cores in a MaxSAT solver. Yet, core reuse leads to a decline in performance in MaxSAT (see Sect. 4). This reveals the inherent issue of computing *cores of poor quality* when solving the smaller instance (see Example 1). In SAT, poor quality clauses from previous computations are eventually deleted. In MaxSAT, poor quality cores can be detrimental to the rest of the computation. This is especially true for the weighted *Fu & Malik* algorithm, which creates new clauses by splitting [19]. Bad quality cores are also known to arise in the standard formulation of weighted MaxSAT. There are approaches to resolve the issue, namely *stratification* [2] and *formula partitioning* [21], which iteratively consider subformulas of the original formula. Note that the same ideas cannot be easily adapted to our setting since the complete MaxSAT formula in our case is not available to the algorithm in advance. Also note that although stratification can be applied to separate MaxSAT instances in the sequence, many of them are satisfiable, which results in stratification being inefficient in practice, as also confirmed by our experience.

The proposed approach uses two levels of incrementality at the same time: (1) it uses incremental SAT calls inside a MaxSAT solver and (2) it makes MaxSAT calls also incremental. This means that all the information learnt during the sequential problem solving is kept until the problem is solved completely. Although the standard way to solve a sequence of MaxSAT instances is to restart a MaxSAT solver at each iteration while doing incremental SAT calls inside, alternatively one could consider using incrementality only for the MaxSAT calls instead. For this, one needs to keep all unsatisfiable cores computed at each preceding MaxSAT call, relax the corresponding clauses of the formula, and reconstruct the cardinality constraints.

Observe that the proposed ideas cannot be easily applied to algorithms that are *not* core-guided. In the classical SAT-UNSAT, UNSAT-SAT linear and binary search MaxSAT algorithms, the SAT solver might learn constraints that are invalid for solving the next MaxSAT formula, so it would have to be restarted for each MaxSAT formula in the sequence. Also note that the *Fu & Malik* algorithm has the drawback of relaxing clauses more than once and thus introducing many auxiliary variables. Therefore, it is of great interest to adapt the proposed ideas to more recent MaxSAT algorithms that resolve this issue (e.g. [1,4–6,8,9,14,15,24–26]). An immediate improvement of the proposed approach would be to devise more fine-grained restart strategies, that is, selectively keeping certain good cores, instead of completely restarting from scratch. Finally, it is also interesting to explore incrementality when clauses are not only added but also deleted.

6   Conclusion

This paper explores an incremental approach to core-guided MaxSAT solving. We begin by extending a core-guided MaxSAT algorithm for sequences of instances where clauses are gradually added. Experimental evaluation shows that this approach in fact yields *worse* performance than applying the MaxSAT solver on each instance from scratch. This is due to the inherent problem of learning "bad" information from instances earlier in the sequence. We propose *restarts* which enable discarding learnt information if deemed unuseful. Our restart strategy significantly outperforms the non-incremental version.

References

 1. Alviano, M., Dodaro, C., Ricca, F.: A MaxSAT algorithm using cardinality constraints of bounded size. In: IJCAI (2015)
 2. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving SAT-Based weighted MaxSAT solvers. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 86–101. Springer, Heidelberg (2012)
 3. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 427–440. Springer, Heidelberg (2009)
 4. Ansótegui, C., Bonet, M.L., Levy, J.: SAT-based MaxSAT algorithms. Artif. Intell. **196**, 77–105 (2013)
 5. Ansótegui, C., Didier, F., Gabàs, J.: Exploiting the structure of unsatisfiable cores in MaxSAT. In: IJCAI (2015)
 6. Bjorner, N., Narodytska, N.: Maximum satisfiability using cores and correction sets. In: IJCAI (2015)
 7. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: A modular approach to MaxSAT modulo theories. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 150–165. Springer, Heidelberg (2013)
 8. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 225–239. Springer, Heidelberg (2011)
 9. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MAXSAT. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 166–181. Springer, Heidelberg (2013)
10. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
11. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. **89**(4), 543–560 (2003)
12. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006)

13. Grigore, R., Yang, H.: Abstraction refinement guided by a learnt probabilistic model. In: POPL (2016)
14. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: AAAI (2011)
15. Ignatiev, A., Morgado, A., Manquinho, V.M., Lynce, I., Marques-Silva, J.: Progression in maximum satisfiability. In: ECAI (2014)
16. Mangal, R., Zhang, X., Kamath, A., Nori, A.V., Naik, M.: Scaling relational inference using proofs and refutations. In: AAAI (2016)
17. Mangal, R., Zhang, X., Nori, A.V., Naik, M.: Volt: a lazy grounding framework for solving very large MaxSAT instances. In: Heule, M., et al. (eds.) SAT 2015. LNCS, vol. 9340, pp. 299–306. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24318-4_22
18. Mangal, R., Zhang, X., Nori, A.V., Naik, M.: A user-guided approach to program analysis. In: FSE (2015)
19. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 495–508. Springer, Heidelberg (2009)
20. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 531–548. Springer, Heidelberg (2014)
21. Martins, R., Manquinho, V.M., Lynce, I.: On partitioning for maximum satisfiability. In: ECAI 2012 (2012)
22. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: a modular MaxSAT solver. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 438–445. Springer, Heidelberg (2014)
23. MaxSAT evaluations. http://www.maxsat.udl.cat/
24. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 564–573. Springer, Heidelberg (2014)
25. Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: a survey and assessment. Constraints **18**(4), 478–534 (2013)
26. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: AAAI (2014)
27. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. **62**(1–2), 107–136 (2006)
28. Zhang, X., Mangal, R., Grigore, R., Naik, M., Yang, H.: On abstraction refinement for program analyses in Datalog. In: PLDI (2014)
29. Zhang, X., Mangal, R., Nori, A.V., Naik, M.: Query-guided maximum satisfiability. In: POPL (2016)

Modelling and Solving Multi-mode Resource-Constrained Project Scheduling

Ria Szeredi[1] and Andreas Schutt[1,2(✉)]

[1] The University of Melbourne, Melbourne, Australia
ria.szeredi@student.unimelb.edu.au
[2] Decision Sciences, Data61, CSIRO, Canberra, Australia
andreas.schutt@data61.csiro.au

Abstract. The resource-constrained project scheduling problem is a fundamental scheduling problem which comprises activities, scarce resources required by activities for their execution, and precedence relations between activities. The goal is to find an optimal schedule satisfying the resource and precedence constraints. These scheduling problems have many applications, ranging from production planning to project management. One of them concerns multi-modes of activities, in which each mode represents a different time-resource or resource-resource trade-off option. In recent years, constraint programming technologies with nogood learning have pushed the boundaries for exact solution methods on various resource-constrained scheduling problems, but, surprisingly, have not been applied on multi-mode resource-constrained project scheduling. In this paper, we investigate different constraint programming models and searches and show the superiority of such technologies in comparison to the current state of the art. Our best approach solved all remaining open instances from a well-established benchmark library.

1 Introduction

The multi-mode resource-constrained project scheduling problem (MRCPSP) is an extension of the well-studied resource-constrained project scheduling problem (RCPSP), which comprises a set of non-preemptive activities, a set of resources with a constant capacity over time, and precedence relations between pairs of activities. For both problems, a start-time schedule for the activities is sought that respects the precedence relations, does not overload a resource at any point in time, while minimising the project duration (makespan). The differences between these problems are that activities can be executed in different modes and resources can be non-renewable in MRCPSP, while activities have a single mode and all resources are renewable in RCPSP. Different modes for an activity model time-resource and resource-resource trade-offs. These scheduling problems are NP-hard and have numerous applications, such as production planning, manufacturing, chemical processing, and project management [16].

An excellent overview of different and state of the art methods can be found in [5]. Most exact solution methods for solving MRCPSP are based on integer programming using branch-and-bound or branch-and-cut. The best methods were

published in [1,16]. Zhu et al. [16] present an exact branch-and-cut algorithm based on integer linear programming (ILP). They apply several pre-processing steps in order to reduce the number of variables in their models, pre-compute cuts from resource conflicts and precedence relations. In addition, a dedicated branching rule is developed for taking the different modes into account. To the best of our knowledge, it is the best exact method so far. Coelho et al. [1] propose a solution approach that decomposes the problem into two sub-problems. The first sub-problem consists of the assignment of the modes of execution solved by a Boolean Satisfiability (SAT) solver. The second sub-problem considers the fixed modes from the first sub-problem and solves the remaining problem using a local search method.

Closely related problems to MRCPSP are RCPSP and MRCPSP with generalised precedence relations (MRCPSP/max). In both cases, the best exact solution methods are based on constraint programming (CP) technologies incorporating nogood learning. For RCPSP, [12,13] present a branch-and-bound approach that is based on lazy clause generation (LCG) [2,7]. LCG is a CP solver that incorporates, amongst others, conflict analysis, conflict-driven search, and unit propagation on conjunction of clauses from SAT solvers. Exact solution approaches based on LCG are the best exact solution methods for various scheduling and packing problems [4,10,11,13–15].

For MRCPSP/max, [8] recently proposed a branch-and-bound approach formulated as a constraint integer program and implemented in the SCIP framework, which also has nogood learning facilities. In order to solve the problem, they implemented two new global constraints for generalised precedence relations and renewable resources, taking the multiple modes of an activity into account. The latter one is an extension of the global constraint `cumulative`. Their method is the best exact solution method for MRCPSP/max.

Surprisingly, no CP technology with nogood learning has been applied to MRCPSP. This paper addresses this gap and not only shows such a method outperforms the state of the art in [16], but also discusses different models and search strategies. In addition, we close all remaining open instances from the well-established benchmark library PSPLib.

2   MRCPSP Model

MRCPSP consists of a *set of non-preemptive activities* $V = \{1, 2, \ldots, n\}$, a *set of precedence relations* $E \subseteq V \times V$, and a *set of resources* $\mathcal{R}$. The set of resources is partitioned into the *set of renewable resources* $\mathcal{R}^R$ and the *set of non-renewable resources* $\mathcal{R}^N$. A resource $k \in \mathcal{R}$ has a discrete *resource capacity* R_k. An activity i has a fixed *set of modes* $\mathcal{M}_i$. For each mode $m \in \mathcal{M}_i$, the activity has a discrete non-negative *duration* (processing time) p_i^m and a discrete non-negative *resource requirement* r_{ik}^m for each resource $k \in \mathcal{R}$ over the planning horizon. The discrete non-negative *start time* S_i and the *mode of execution* M_i must be determined by the solution approach. The planning horizon starts at time period 0.

Definition 1 (Solution of MRCPSP). *A solution of MRCPSP is an assignment of start times S_i and modes of executions M_i for each activity i such that the following constraints hold*

$$\forall i \in V : 0 \leq S_i \quad \wedge \quad M_i \in \mathcal{M}_i$$

$$\forall (i,j) \in E : S_i + p_i^{M_i} \leq S_j \tag{1}$$

$$\forall k \in \mathcal{R}^R, \forall \tau \geq 0 : \sum\nolimits_{i \in V : S_i \leq \tau < S_i + p_i^{M_i}} r_{ik}^{M_i} \leq R_k \tag{2}$$

$$\forall k \in \mathcal{R}^N : \sum\nolimits_{i \in V} r_{ik}^{M_i} \leq R_k \tag{3}$$

where constraint (1) ensures the satisfaction of the precedence constraints and constraints (2–3) respectively guarantee a non-overload of renewable and non-renewable resources. An optimal solution additionally minimises the project duration (makespan), i.e., $\min \max_{i \in V}(S_i + p_i^{M_i})$.

2.1 Solver Independent Model

For the sake of readability, we present a "simplified" model and the "user-defined" searches using the solver-independent modelling language MiniZinc [6].

An MRCPSP instance is represented by the following parameters, whose meaning is given in the comment next to them where the arrays `mact` and `mode` respectively map a mode to its activity and an activity to its set of modes.

```
set of int: Res;      % Set of resources
set of int: Act;      % Set of activities
set of int: Mod;      % Set of modes
array[Res] of int: rcap;      % Resource capacity
array[Res] of int: rtype;     % Resource type (1: renewable; 2: non-renewable)
set of int: RRes = {k | k in Res where rtype[k] = 1};
set of int: NRes = {k | k in Res where rtype[k] = 2};
array[Mod] of Act: mact;        % Corresponding activity of a mode
array[Mod] of int: mdur;        % Duration of modes
array[Res,Mod] of int: mrreq;   % Resource requirements of modes
array[Act] of set of Mod: mode = [{m | m in Mod where mact[m] = i} | i in Act
    ]; % Set of modes for each activity
array[Act] of set of Act: succ; % Set of successors
```

Variables. Three variables are created for each activity `i` reflecting its start time `start[i]`, its duration `adur[i]`, and its resource requirements `arreq[k,i]` for each resource `k`. The duration and resource requirements are determined by the mode of execution. A Boolean variable `mrun[m]` models whether the mode `m` is executed in the final schedule. Lastly, the objective variable is defined as `makespan`.

```
array[Mod] of var bool: mrun;
array[Act] of var 0..UB: start;
array[Act] of var int: adur = [let {var {mdur[m] | m in mode[i]}: x} in x | i
    in Act];
array[Res,Act] of var int: arreq = array2d(Res,Act,[let {var {mrreq[k,m] | m
    in mode[i]}: x} in x | k in Res, i in Act]);
var 0..UB: makespan;
```

The variables in `start` and the variable `makespan` have an initial domain `0..UB` where `UB` is the initial upper bound on the objective. Unless otherwise stated, `UB` is initialised by `sum(i in Act)(max([mdur[m] | m in mode[i]]));`.

Activities and mode constraints. The duration and resource requirements of an activity are linked via a set of linear constraints, encapsulated in the first two constraints below. The last constraint ensures that exactly one mode is executed for each activity.

```
constraint forall(i in Act)(adur[i] = sum(m in mode[i])(mdur[m] * mrun[m]));
constraint forall(i in Act, k in Res)(arreq[k,i] = sum(m in mode[i])(mrreq[k,
    m] * mrun[m]));
constraint forall(i in Act)(sum(m in mode[i])(mrun[m]) = 1);
```

Alternatively, we can create an auxiliary variable `mi` and replace the first two constraints above by element constraints (elem) for modelling `adur` and `arreq`.

```
constraint forall(i in Act)( let {var mode[i]: mi} in (mrun[mi] = 1 /\ adur[i
    ] = mdur[mi] /\ forall(k in Res)(arreq[k,i] = mrreq[k,mi])) );
```

Precedence constraints. The precedence constraints are modelled as usual using the start time and duration variables.

```
constraint forall(i in Act, j in succ[i])(start[i] + adur[i] <= start[j]);
```

Renewable resource constraints. There are two options for modelling renewable resources using the global constraint cumulative. The first option (ract) creates one activity in the cumulative constraint for each activity, in which the durations and resource requirements are variables.

```
constraint forall(k in RRes)(cumulative(start, adur, [arreq[k,i] | i in Act],
    rcap[k]));
```

The second option (rmode) creates one activity for each mode, resulting in a greater number of activities generated in the cumulative constraint, but having only the resource requirements as variables. These variables can be created as a variable view on the Boolean variables in `mrun`.

```
constraint forall(k in RRes)(cumulative([start[mact[m]] | m in Mod], mdur, [
    mrreq[k,m] * mrun[m] | m in Mod], rcap[k]));
```

On the one hand, a cumulative propagator can exploit the knowledge of the duration-resource-requirement pairing in rmode. On the other hand, it loses the knowledge that exactly one mode has to be executed. Note that [8] extended the cumulative propagator for taking multi-modes for activities into account. However, the considered solvers in this paper do not provide this extension.

Non-renewable resource constraints. Non-renewable resources are simply modelled by linear constraints. As for the renewable resource constraints, there are again two options. The first option (nact) models it via the activities using the variables for the resource requirements,

```
constraint forall(k in NRes)(sum(i in Act)(arreq[k,i]) <= rcap[k]);
```

whereas the second option (nmode) models via the modes using the Boolean variables mrun.

```
constraint forall(k in NRes)(sum(m in Mod)(mrreq[k,m] * mrun[m]) <= rcap[k]);
```

Pairwise non-overlapping constraints. Pairwise non-overlapping constraints for activities might speed up the solution process and be advantageous for learning solvers. These are redundant constraints with respect to the renewable resource constraints. Two activities i and j cannot be run concurrently and are disjunct iff $\forall m_i \in \mathcal{M}_i, \forall m_j \in \mathcal{M}_j, \exists k \in \mathcal{R}^R : r_{ik}^{m_i} + r_{jk}^{m_j} > R_k$. The next constraint (disj) ensures that one of such activities is run before the other one.

```
predicate post_noc_disj(int: i, int: j) = (start[i] + adur[i] <= start[j] \/
    start[j] + adur[j] <= start[i]);
```

Other pairs of activities that might be a disjunct in some modes are modelled by following half-reified constraints.

```
predicate post_noc_mode(int: i, int: j) = forall(mi in mode[i], mj in mode[j]
    where exists(k in RRes)(mrreq[k,mi] + mrreq[k,mj] > rcap[k]))( (mrun[mi]
    /\ mrun[mj]) -> (start[i] + mdur[mi] <= start[j] \/ start[j] + mdur[mj]
    <= start[i]) );
predicate post_noc_rres(int: i, int: j) = forall(k in RRes)( (arreq[k,i] +
    arreq[k,j] > rcap[k]) -> (start[i] + adur[i] <= start[j] \/ start[j] +
    adur[j] <= start[i]) );
```

The predicate post_noc_mode (nocm) models non-overlapping constraints for each mode pair, while post_noc_rres (nocr) only for each renewable resource.

Objective constraints. The objective variable is constrained by the latest end time of an activity as follows.

```
constraint makespan = max(i in Act where succ[i]={})(start[i] + adur[i]);
constraint forall(i in Act where succ[i]={})(start[i] + adur[i] <= makespan);
```

Search strategies. We investigated different search strategies including the default ones of the considered solvers.

```
ann: mode_s    = bool_search(mrun, input_order, indomain_max, complete);
ann: start_s   = int_search(start, smallest, indomain_min, complete);
ann: adur_s    = int_search(adur, smallest, indomain_min, complete);
ann: arreq_s   = int_search([arreq[k,i] | k in NRes, i in Act], smallest,
    indomain_min, complete);
ann: modeThenStart         = seq_search([ mode_s, start_s ]);
ann: arreqThenMode         = seq_search([ arreq_s, modes_s ]);
ann: arreqThenStart        = seq_search([ arreq_s, start_s ]);
ann: arreqThenModeThenStart = seq_search([ arreq_s, mode_s, start_s ]);
ann: durThenStart          = seq_search([ adur_s, modes_s ]);
ann: durThenModeThenStart  = seq_search([ adur_s, mode_s, start_s ]);
```

The search modeThenStart splits the search into two stages. First, it assigns the mode to each activity and then solves the remaining RCPSP by searching on the start times. The next three searches arreqThenMode, arreqThenStart and arreqThenModeThenStart assign the smallest resource requirements of activities for non-renewable resources first, before continuing the search on the modes and/or the start times. The last two searches assign the shortest duration of each activity before assigning the mode and/or the start times. Note that if a search does not assign all variables, then the solver uses its default search to assign the remaining variables.

3   Experiments

We conducted experiments on the well-studied MRCPSP benchmark set from
the PSPLib available at www.om-db.wi.tum.de/psplib/. The benchmark set con-
tains different test sets, which differ in their characteristics. Except for the test
set `j30`, all instances are closed. In the remainder of this paper, we concen-
trate on `j20` and `j30`. Instances from these test sets are composed of 20 and
30 activities having between one and three modes, two renewable resources, two
non-renewable resources, and a number of precedence relations.

All experiments were run on machines operating CentOS 6.5 with AMD 6-
Core Opteron 4334 clocking 3.1 GHz, and 64 GB memory. A runtime limit of
10 min was imposed unless otherwise stated. For compiling the MiniZinc model
into the solver-specific FlatZinc format, we used MiniZinc 2.0.13 downloaded
from www.minizinc.org. The following CP solvers were investigated Gecode 4.4.1
(`gecode`), Opturion CPX 1.0.2 (`ocpx`), G12/LazyFD (`lazyfd`), and Chuffed rev.
885 (`chuffed`) where the last three are LCG solvers.

3.1   Comparison of Models

Section 2.1 presents two ways of modelling renewable resource, non-renewable
resource and pairwise non-overlapping constraints. Since the behaviour of non-
learning and learning solvers can be significantly different, we present the results
for `gecode` and `chuffed` as representatives for each kind.

Table 1 shows the results for both solvers using the search `modeThenStart` in
two parts. Part I shows the different combinations for the renewable (rres) and
non-renewable (nres) resource constraints. For both solvers, the best combination
in terms of number of optimal solutions (#opt), mean runtime in seconds (m.rt.),
and mean number of explored nodes (m.#nodes) is to use the activity represen-
tation (act) for both constraints. Interestingly, `chuffed` performance drastically
decays when using the mode representation (`nmode`) for non-renewable resource
constraints, while `gecode` only worsens slightly. This could indicate that `chuffed`
misses some propagation. Note that we also run the experiments with both activ-
ity and mode representations, but the runtime increased substantially while the
number of explored nodes did not change significantly.

Part II in Table 1 lists the results when the redundant non-overlapping con-
straints are used. In each setting, it is advantageous to use the redundant con-
straints. The best option is to use constraints for non-overlapping (`disj`, `nocr`)
and activity representations (`ract`, `nact`), which gives the lowest mean runtimes
for both solvers and the highest number of optimally solved instances in the case
for `gecode`. Part III shows that using element constraints (`elem`) for the activity
and mode constraints performs similar to the best option in Part II. For the
remainder, we consider the model in Part III, which performs at best on the test
set `j30`.

Table 1. Comparison of different models on 554 instances from test set `j20`.

part	rres	nres	disj	nocm	nocr	elem	chuffed #opt	m.rt.	m.#nodes	gecode #opt	m.rt.	m.#nodes
I	ract	nact					554	1.36s	21k	469	99s	6615k
	ract	nmode					525	56.7s	1544k	457	134s	9577k
	rmode	nact					554	2.41s	40k	464	104s	3509k
	rmode	nmode					516	61.2s	1421k	435	152s	5444k
II	ract	nact	✓				554	1.15s	19k	472	98s	6610k
	ract	nact	✓	✓			554	2.03s	18k	477	94s	4388k
	ract	nact	✓		✓		554	1.15s	13k	478	92s	5495k
III	ract	nact	✓		✓	✓	554	1.16s	13k	478	92s	5139k

Table 2. Comparison of different search strategies on test set `j30`.

search	chuffed #opt	#feas	#unsat	#unkn	m.rt	m.#nodes
`modeThenStart`	495	57	88	0	60.3 s	615 k
`arreqThenMode`	488	59	88	5	64.0 s	549 k
`arreqThenStart`	491	61	88	0	61.7 s	580 k
`arreqThenModeThenStart`	490	62	88	0	62.4 s	614 k
`durThenStart`	506	46	88	0	56.3 s	628 k
`durThenModeThenStart`	506	46	88	0	58.3 s	682 k

3.2   Comparison of Search Strategies

Table 2 presents the outcome of the different searches when the best model (see previous sub-section) is used. Searches starting with the assignment of duration variables (`durThenStart` and `durThenModeThenStart`) are the quickest and optimally solve the greatest number of instances. The next best search is `modeThenStart`, while the remaining searches, all of which assign the resource requirement variables for non-renewable resources first, perform worst. Interestingly, searching over mode variables is slightly worse than leaving them out. For instance, the mean runtime of `durThenStart` is slightly less than that of `durThenModeThenStart`. Similar results for the search strategies were obtained on the models presented in Table 1 in preliminary experiments. For `chuffed`, we also ran each search in combination with `chuffed` activity based search, in which `chuffed` alternates between the two searches at each restart. The results are similar, but with the alternating searches more instances were solved to optimality and the mean runtime and number of nodes were lower.

3.3   Comparison of Solvers

Table 3 shows the results of the different solvers for the best model in combination with the solver's default search and the best search. Clearly, the default searches,

Table 3. Comparison of the different solvers on the test set `j30`.

solver	search	#opt	#feas	#unsat	#unkn	m.rt	m.#nodes
`gecode`	`durThenStart`	422	105	40	73	184 s	6056 k
`gecode`	`default`	385	87	0	168	246 s	6184 k
`ocpx`	`durThenStart`	468	84	44	44	151 s	>52 k
`ocpx`	`default`	492	58	88	2	113 s	>33 k
`lazyfd`	`durThenStart`	473	78	11	78	166 s	n/a
`lazyfd`	`default`	515	37	88	0	53.1 s	n/a
`chuffed`	`durThenStart`	506	46	88	0	56.3 s	628 k
`chuffed`	`default`	540	12	88	0	18.6 s	148 k

which are conflict driven, of the nogood learning solvers drastically outperform
the user-defined searches. Note that `chuffed` default search alternates between a
conflict driven and the user-defined search. As expected, nogood learning solvers
outperform `gecode`, because their derived nogoods avoid the re-exploration of
similar search sub-trees proven to be infeasible and information retrieved by
the conflict analysis is used to guide the search. The clear winner is `chuffed`.
The big difference in the performance of the LCG solvers may be surprising
at first, but can be explained by the differences in the cumulative constraints.
All three LCG solvers implement the cumulative constraint using the time-table
propagation from [13], but only `lazyfd` and `chuffed` allow for variable durations
and resource requirements as input as described in [9]. In addition, MiniZinc
does not provide an interface for the cumulative constraint of `lazyfd`. Hence,
the cumulative constraint is mapped into the time-indexed decomposition when
compiling the model for `ocpx` and `lazyfd`.

3.4   Comparison to the State of the Art

Zhu et al. [16] present—to the best of our knowledge—the best exact solution
method for MRCPSP. This method is based on Integer Linear Programming
using a branch-and-cut for minimizing the makespan. It is implemented in the
Mixed Integer Programming solver CPLEX 7.5. They run their method on a
Linux machine with 1.8 GHz Xeon processor. Within one hour, it could optimally
solve 506 instances out of 552 feasible instances with a mean total runtime
of 393.1 s. The average runtime was 125.25 s for finding the best solution. By
constrast, the best set up of `chuffed` could optimally solve 540 instances (34
more) within 10 min. In addition, the runtimes of `chuffed` are drastically lower.
Thus, `chuffed` outperforms the state of the art.

Closed instances. With respect to [5,16], there are 46 open instances in the
test set `j30`. Within the 10 min runtime limit, the presented solver was able to
close 34 of them. In preliminary experiments, we ran `chuffed` without a runtime

limit and were able to close all instances. The last instance was closed after 18 hours.

4   Conclusion

We investigated different CP models for solving MRCPSP. To our best knowledge, it is the first published CP model, on which an exact solution method with nogood learning was applied. The best model uses the activity representation for modelling the resource constraints via the constraint `cumulative` and pairwise non-overlapping constraints for activities that might be in disjunction. All the considered user-defined searches were inferior to the default search of the CP solvers. The LCG solver `chuffed` was the best performing solver, which also outperformed the state of the art ILP solver [16]. Within 10 min, all open instances were closed except 12. Relaxing the time limit, all remaining open instances were closed within 18 h.

References

1. Coelho, J., Vanhoucke, M.: The Multi-mode resource-constrained project scheduling problem. In: Schwindt, C., Zimmermann, J. (eds.) Handbook on Project Management and Scheduling. International Handbooks on Information Systems, vol. 1, pp. 491–511. Springer, Heidelberg (2015)
2. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent [3], pp. 352–366
3. Gent, I.P. (ed.): Principles and Practice of Constraint Programming - CP 2009. LNCS, vol. 5732. Springer, Heidelberg (2009)
4. Kreter, S., Schutt, A., Stuckey, P.J.: Modeling and solving project scheduling with calendars. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 262–278. Springer, Heidelberg (2015)
5. Mika, M., Waligóra, G., Węglarz, J.: Overview and state of the art. In: Schwindt, C., Zimmermann, J. (eds.) Handbook on Project Management and Scheduling. International Handbooks on Information Systems, vol. 1, pp. 445–490. Springer, Heidelberg (2015)
6. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.R.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)
7. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
8. Schnell, A., Hartl, R.F.: On the efficient modeling and solution of the multi-mode resource-constrained project scheduling problem with generalized precedence relations. OR Spectrum **38**(2), 283–303 (2015)
9. Schutt, A.: Improving scheduling by learning. Ph.D. thesis, The University of Melbourne (2011). http://hdl.handle.net/11343/36701
10. Schutt, A., Chu, G., Stuckey, P.J., Wallace, M.G.: Maximising the net present value for resource-constrained project scheduling. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 362–378. Springer, Heidelberg (2012)

11. Schutt, A., Feydy, T., Stuckey, P.J.: Scheduling optional tasks with explanation. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 628–644. Springer, Heidelberg (2013)
12. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Why cumulative decomposition is not as bad as it sounds. In: Gent [3], pp. 746–761
13. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints **16**(3), 250–282 (2011)
14. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving RCPSP/max by lazy clause generation. J. Sched. **16**(3), 273–289 (2012)
15. Schutt, A., Stuckey, P.J., Verden, A.R.: Optimal carpet cutting. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 69–84. Springer, Heidelberg (2011)
16. Zhu, G., Bard, J.F., Yu, G.: A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. INFORMS J. Comput. **18**(3), 377–390 (2006)

A Nearly Exact Propagation Algorithm for Energetic Reasoning in $\mathcal{O}(n^2 \log n)$

Alexander Tesch$^{(\boxtimes)}$

Zuse Institute Berlin, Takustrasse 7, 14195 Berlin, Germany
`tesch@zib.de`

Abstract. In constraint programming, energetic reasoning constitutes a powerful start time propagation rule for cumulative scheduling problems (CuSP). This article first presents an improved time interval checking algorithm that is derived from a polyhedral model. In a second step, we extend this algorithm to an energetic reasoning propagation algorithm with time complexity $\mathcal{O}(n^2 \log n)$ where n denotes the number of jobs. The idea is based on a new sweep line subroutine that efficiently evaluates energy overloads of each job on the relevant time intervals. In particular, our algorithm performs energetic reasoning propagations for every job. In addition, we show that on the vast number of relevant intervals our approach achieves the maximum possible propagations according to the energetic reasoning rule.

1 Introduction

The *cumulative scheduling problem* (CuSP) considers a set of jobs J where each job $j \in J$ is given a processing time $p_j \in \mathbb{Z}_{>0}$, a resource demand $d_j \in \mathbb{Z}_{>0}$ and a scheduling interval $[e_j, l_j] \subset \mathbb{R}$. For every job $j \in J$ we want to compute start times $s_j \in [e_j, l_j - p_j]$ such that at any point in time the resource consumption of all jobs does not exceed the maximum capacity $D \in \mathbb{Z}_{>0}$. Equivalently, the CuSP can be described as the feasibility problem

$$\text{find} \qquad s \in \mathbb{R}^n$$

$$\text{such that} \sum_{j \in J: s_j \le t < s_j + p_j} d_j \le D \qquad \forall t \in \mathbb{R} \tag{1}$$

$$e_j \le s_j \le l_j - p_j \qquad \forall j \in J. \tag{2}$$

Usually, CuSP feasibility tests are subroutines for more specific scheduling problems such as *makespan minimization* [8,9]. A CuSP checks if a current subproblem may lead to feasible solution of the main problem. If this is not the case, the search tree can be pruned.

In order to solve the CuSP the literature proposes domain branching in combination with specific feasibility and propagation algorithms. The most

common ones are *Time-Tabling* [6,9,15], *Edge-Finding* [7,13], *Extended Edge-Finding* [14], *Time-Table Edge-Finding* [8,11,12], *Energetic Reasoning* [5] and *Not-First/Not-Last Pruning* [10].

This paper focuses on energetic reasoning. Except for the last, energetic reasoning dominates all of the stated rules. In practice, however, the weaker but faster propagation rules are preferred over energetic reasoning due to its high complexity of $\mathcal{O}(n^3)$, see Baptiste et al. [5]. Therefore, it is natural to ask for faster implementations of energetic reasoning. In this context, some approaches aim to improve the energetic reasoning rule directly or its external conditions. Berthold, Heinz and Schulz [4] characterize time intervals where energetic reasoning cannot detect infeasibility. Such intervals can be neglected in the standard energetic reasoning algorithm which leads to a reasonable computation time improvement. Similarly, Derrien et al. [3] sharpen the original characterization of relevant time intervals of Baptiste et al. [5] and apply the standard energetic reasoning algorithm to the reduced set of intervals. Both approaches are *exact*, that is they compute the maximum energetic reasoning propagations for all jobs. Their complexity is $\mathcal{O}(n^3)$. Recently, Bonifas [2] presents a new $\mathcal{O}(n^2 \log n)$ algorithm that computes one beneficial job for each relevant interval and propagates energetic reasoning on it. Unlike previous approaches, his algorithm detects at least one possible energetic reasoning propagation. But in general, the total number of propagations is $\mathcal{O}(1)$. Thus, to ensure propagations for all jobs his algorithm generally needs to be executed $\mathcal{O}(n)$ times which gives a complexity of $\mathcal{O}(n^3 \log n)$. However, the method seems practically relevant, if one focuses on one job propagations [2]. The idea of Bonifas' algorithm is based on a geometric interpretation of energetic reasoning which leads to upper envelope computations of piecewise linear functions in the plane.

In this article, we proceed from a related geometric interpretation. In contrast to Bonifas [2], we state a different geometric problem which allows us to compute energetic reasoning propagations for all jobs instead of only one. Hence our approach yields $\mathcal{O}(n)$ more propagations in general. In addition, for each propagation that is found in [2] our algorithm finds an equal or stronger propagation. The complexity of our algorithm is $\mathcal{O}(n^2 \log n)$ compared to the complete $\mathcal{O}(n^3 \log n)$ algorithm in [2]. Hence, our algorithm supersedes Bonifas' algorithm. Compared to the exact $\mathcal{O}(n^3)$ approach of Derrien et al. [3], our algorithm does not provably compute the maximum energetic reasoning propagations. But we show that our approach yields maximum propagations on the huge majority of relevant intervals. We give a precise characterization of the corresponding interval set.

Our paper is organized as follows. In Sect. 2 we introduce the concepts of energetic reasoning. Section 3 gives an alternative characterization for the set of relevant time intervals from polyhedral theory. From this, we deduce an improved interval checking algorithm in Sect. 4. Furthermore, we extended this algorithm by energetic reasoning propagations in Sect. 5. Its basis forms a sweep line subroutine that is introduced in Sect. 6. In Sect. 7 we characterize the set of time intervals where our algorithm performs maximum propagations. Section 8

compares our methods to the current state of the art. Finally, we conclude our results in Sect. 9.

2 Energetic Reasoning

In the following we introduce the basic concepts of energetic reasoning for CuSP, see Baptiste et al. [5]. Assume a CuSP instance as introduced in Sect. 1. For each job $j \in J$ define

$$\mu_j(t_1, t_2) = \min\{p_j, t_2 - t_1, \max\{0, e_j + p_j - t_1\}, \max\{0, t_2 - l_j + p_j\}\} \quad (3)$$

as the *minimum left-/right-shift duration* in the time interval $[t_1, t_2] \subset \mathbb{R}$. Moreover, define the *energy overload* in the time interval $[t_1, t_2] \subset \mathbb{R}$ as

$$\omega(t_1, t_2) = \sum_{j \in J} d_j \cdot \mu_j(t_1, t_2) - D \cdot (t_2 - t_1) \quad (4)$$

which is the slack between the *consumed energy* and the *available energy* in a time interval $[t_1, t_2]$. If in any time interval $[t_1, t_2] \subset \mathbb{R}$ the consumed energy exceeds the available energy, that is $\omega(t_1, t_2) > 0$, then the CuSP is infeasible. The time interval of maximum energy overload can be computed in $\mathcal{O}(n^2)$, see [5].

Besides checking infeasibility, energetic reasoning reduces the variable domain which consists of scheduling intervals $[e_j, l_j]$ for all jobs $j \in J$. Assume a job $j \in J$ is *left-shifted*, that is $s_j = e_j$, then

$$\mu_j^{left}(t_1, t_2) = \max\{0, \min\{t_2, e_j + p_j\} - \max\{t_1, e_j\}\} \quad (5)$$

defines the *left-shift duration* of job j in the interval $[t_1, t_2] \subset \mathbb{R}$. In addition,

$$\omega_j(t_1, t_2) = \omega(t_1, t_2) + d_j \cdot (\mu_j^{left}(t_1, t_2) - \mu_j(t_1, t_2)) \quad (6)$$

denotes the overload in the interval $[t_1, t_2] \subset \mathbb{R}$ that occurs if job $j \in J$ is left-shifted. The energetic reasoning propagation rule states: if there is an energy overload in the interval $[t_1, t_2] \subset \mathbb{R}$ due to left-shifting job $j \in J$, that is $\omega_j(t_1, t_2) > 0$, then e_j is an invalid earliest start time and thus can be delayed.

Theorem 1 (Baptiste et al. [5]). *Given a job $j \in J$ and a time interval $[t_1, t_2] \subset \mathbb{R}$ with $\omega_j(t_1, t_2) > 0$ then the earliest start time e_j can be updated to*

$$e_j = t_2 - \mu_j(t_1, t_2) + \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil. \quad (7)$$

Note that the *right-shift* case is equivalent to the left-shift case by symmetry at time $t = 0$, see Derrien et al. [3]. Since there are $\mathcal{O}(n^2)$ relevant time intervals [5], the standard energetic reasoning algorithm checks Theorem 1 for all jobs on all relevant time intervals which yields an exact $\mathcal{O}(n^3)$ energetic reasoning propagation algorithm. The currently tightest characterization of the $\mathcal{O}(n^2)$ time intervals is given by Derrien et al. [3].

3   The Energetic Reasoning Polyhedron

Derrien et al. [3] characterize a set of relevant intervals that are sufficient for overload checking. However, they do not state an algorithm that reduces to their characterization. In this section, we introduce a polyhedral model from which we derive an alternative characterization for the same set of relevant time intervals. But our polyhedral model enables us to construct an improved overload checking algorithm that considers a subset of intervals than the algorithm stated in [3].

First, we model the problem of computing an interval $[t_1, t_2] \subset \mathbb{R}$ of maximum overload (4) by a simple linear program. Therefore, let $t_1, t_2 \in \mathbb{R}$ be continuous variables that represent the interval limits. In addition, the variables $\tilde{\mu}_j \geq 0$ model the piecewise linear expression $\mu_j(t_1, t_2)$, as given in (3), for all jobs $j \in J$. Then for any job subset $S \subseteq J$ with $S \neq \emptyset$ define the linear program

$$\max \sum_{j \in I} d_j \cdot \tilde{\mu}_j - D \cdot (t_2 - t_1)$$

$$\tilde{\mu}_j \leq p_j \qquad \forall j \in S \tag{8}$$
$$\tilde{\mu}_j \leq t_2 - t_1 \qquad \forall j \in S \tag{9}$$
$$\tilde{\mu}_j \leq e_j + p_j - t_1 \qquad \forall j \in S \tag{10}$$
$$\tilde{\mu}_j \leq t_2 - l_j + p_j \qquad \forall j \in S \tag{11}$$
$$t_1 \leq t_2 \tag{12}$$
$$\tilde{\mu}_j \geq 0 \qquad \forall j \in S \tag{13}$$

$$t_1, t_2 \in \mathbb{R}$$

in $|S| + 2$ variables. With respect to (4), the objective function maximizes the energy overload in the variable interval $[t_1, t_2] \subset \mathbb{R}$, whose maximum energy is bounded by inequalities (8)–(11) for each variable $\tilde{\mu}_j$ with $j \in S$ according to (3). We define the associated polyhedron of inequalities (8)–(13) by

$$P_S = \{(t_1, t_2, \tilde{\mu}) \in \mathbb{R}^{|S|+2} \mid (t_1, t_2, \tilde{\mu}) \text{ satisfies } (8) - (13)\} \tag{14}$$

which we call the *energetic Reasoning polyhedron* for the job subset $S \subseteq J$.

Lemma 1. *There exists a job subset $S \subseteq J$ such that the maximum overload $\omega^* = \max_{t_1 < t_2} \omega(t_1, t_2)$ equals the optimal objective value of the linear program*

$$\max \sum_{j \in J} d_j \cdot \tilde{\mu}_j - D \cdot (t_2 - t_1), \ (t_1, t_2, \tilde{\mu}) \in P_S.$$

Proof. Let $(t_1^*, t_2^*) \in \mathbb{R}^2$ be the time interval of maximum overload and let $S = \{j \in J \mid \mu_j(t_1^*, t_2^*) > 0\}$. The optimal values $\tilde{\mu}_j^*$ are attained at the minimum right hand side of inequalities (8)–(11), that is $\tilde{\mu}_j^* = \mu_j(t_1^*, t_2^*)$ for all $j \in J$.   □

In the following we characterize the vertices of P_S as they identify intervals of maximum overload. Without loss of generality, we restrict ourselves to vertices $(t_1, t_2, \tilde{\mu}) \in P_S$ with $\tilde{\mu}_j > 0$ for all $j \in S$. Otherwise, if $\tilde{\mu}_j = 0$ for any $j \in S$ then job j does not contribute to the objective function, so we can equivalently consider $P_{S'}$ with $S' = S \setminus \{j\}$. In the following we abbreviate notation and write $P_j = P_{\{j\}}$ and $P_{i,j} = P_{\{i,j\}}$.

Lemma 2. *Let $S \subseteq J$ be a job subset with $S \neq \emptyset$ and $P_S \neq \emptyset$. In addition, let $(t_1, t_2, \tilde{\mu}) \in P_S$ be a vertex of P_S with $\tilde{\mu}_j > 0$ for all $j \in S$. Then either one of the following holds:*

(i) There is a job $j \in S$ such that $(t_1, t_2, \tilde{\mu}_j)$ is a vertex of P_j.
(ii) There are two distinct jobs $i, j \in S$ such that $(t_1, t_2, \tilde{\mu}_i, \tilde{\mu}_j)$ is the intersection of one edge of P_i and one edge of P_j and thus a vertex of $P_{i,j}$.

In order to determine the time interval of maximum energy overload, Lemma 2 allows us to restrict to vertices of P_j or $P_{i,j}$ for dedicated jobs $i, j \in S$ with $i \neq j$. Since the vertices of case (i) and (ii) correspond to the intersection of edges of P_j, or P_i and P_j respectively, Lemma 2 motivates to project the edges of the polyhedron P_j with $j \in J$ to the (t_1, t_2)-plane. The next lemma gives a similar geometric interpretation as presented in Artigues et al. [1].
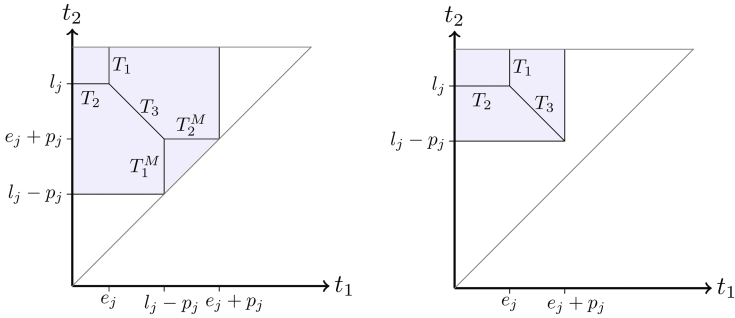
Fig. 1. Two possible shapes of the projected polyhedron P_j (here with lower and upper bounds for t_1 and t_2) with mandatory part (left) or without mandatory part (right).

Lemma 3. *Given a job $j \in J$ and assume the projection of the polyhedron P_j to the (t_1, t_2)-plane. The projected line segments of the edges of P_j that contain a vertex $(t_1, t_2, \tilde{\mu}_j)$ of P_j with $\tilde{\mu}_j > 0$ are given by*

$$T_1(j) = \{(e_j, t_2) \in \mathbb{R}^2 \mid t_2 \geq l_j\}$$
$$T_2(j) = \{(t_1, l_j) \in \mathbb{R}^2 \mid t_1 \leq e_j\}$$
$$T_3(j) = \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 + t_2 = e_j + l_j, \; e_j \leq t_1 \leq \min\{e_j + p_j, l_j - p_j\}\}$$
$$T_1^M(j) = \{(l_j - p_j, t_2) \in \mathbb{R}^2 \mid l_j - p_j \leq t_2 \leq e_j + p_j\}$$
$$T_2^M(j) = \{(t_1, e_j + p_j) \in \mathbb{R}^2 \mid l_j - p_j \leq t_1 \leq e_j + p_j\}.$$

We say job $j \in J$ has a *mandatory part*, if it holds $l_j - p_j < e_j + p_j$. Let $J^M = \{j \in J \mid l_j - p_j < e_j + p_j\}$ denote the set of jobs with mandatory part. Consider Fig. 1. From Lemma 3 we deduce that for any job $j \in J$ the polyhedron P_j can have two combinatorial types: if j has a mandatory part or if j has no mandatory part. If job j has a mandatory part then inequality (9) cuts P_j which yields the additional line segments $T_1^M(j)$ and $T_2^M(j)$. If job j has no mandatory part inequality (9) is dominated by inequalities (8), (10) and (11) which implies $T_1^M(j) = T_2^M(j) = \emptyset$. Combining Lemmas 2 and 3, define for any two jobs $i, j \in J$ with $i \neq j$ the line segment intersection points $\mathcal{T}_j = \{(e_j, l_j)\}, \mathcal{T}_j^M = \{(l_j - p_j, e_j + p_j)\}, \mathcal{T}_{ij} = (T_1(i) \cup T_1^M(i)) \cap (T_2(j) \cup T_2^M(j))$ and $\mathcal{T}'_{ij} = (T_1(i) \cup T_1^M(i) \cup T_2(i) \cup T_2^M(i)) \cap T_3(j)$.

Thus, the set of *relevant time intervals* results as the union

$$\mathcal{T} = \bigcup_{j \in J} \mathcal{T}_j \cup \bigcup_{j \in J^M} \mathcal{T}_j^M \cup \bigcup_{i,j \in J: i \neq j} \mathcal{T}_{ij} \cup \bigcup_{i,j \in J: i \neq j} \mathcal{T}'_{ij} \tag{15}$$

which forms a set of points in the plane.

Theorem 2. *If $(t_1, t_2) \in \mathbb{R}^2$ is a time interval of maximum energy overload then it holds $(t_1, t_2) \in \mathcal{T}$.*

Note that the interval set $\mathcal{T}$ is equivalent to the characterization of Derrien et al. [3]. However, in the next section we derive an improved overload checking algorithm from this characterization of projected line segments.

4 Dynamic Overload Checking Algorithm

In this section we introduce an improved overload checking algorithm that considers a subset of intervals than the checker presented in [3], see Algorithm 1 in the appendix. We modify the $\mathcal{O}(n^2)$ overload checking algorithm of Baptiste et al. [5]. The basic algorithm iterates over all t_1 values of vertical line segments $T_1(i) \cup T_1^M(i)$ with $i \in J$ and over all t_2 values of horizontal and diagonal line segments $T_2(j) \cup T_2^M(j) \cup T_3(j)$ with $j \in J$. At each pair (t_1, t_2) we check for an energy overload $\omega(t_1, t_2) > 0$ which implies infeasibility (line 9).

While the overload checker of [3] iterates all possible t_2 values, our algorithm uses a dynamic list to store only those t_2 values that intersect with the current vertical t_1 line segment. In this context, the following fact is crucial: for any current t_1 value, there is at most one intersecting segment of either $T_2(j), T_2^M(j)$ or $T_3(j)$ for every job $j \in J$, see Fig. 1. For non-decreasing t_1 values they appear in the sequence of either $T_2(j) \rightarrow T_3(j) \rightarrow T_2^M(j)$ or $T_2(j) \rightarrow T_3(j)$ depending on whether it holds $j \in J^M$ or $j \in J \setminus J^M$. Whenever we traverse the intersection point $(e_j, l_j) \in \mathbb{R}^2$ of the line segments $T_2(j)$ and $T_3(j)$ we delete the current $T_2(j)$ line segment and add the $T_3(j)$ line segment to the list (lines 13–16). Analogously, if $j \in J^M$ and we traverse the intersection point $(l_j - p_j, e_j + p_j)$ of the line segments $T_3(j)$ and $T_2^M(j)$ we delete the $T_3(j)$ line segment and add the $T_2^M(j)$ line segment to the list (lines 17–19). Moreover, if we detect a t_2 value
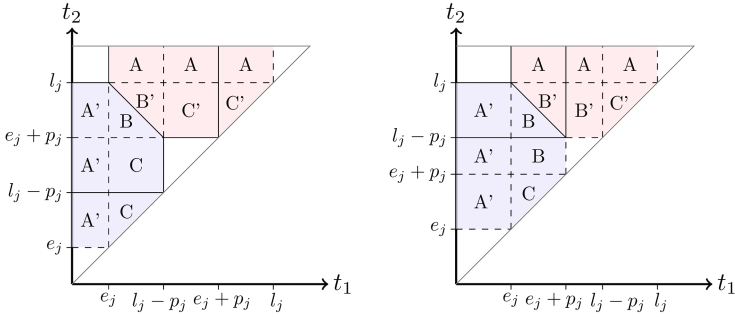
Fig. 2. Figure 1 continued: The slack regions between the left-shift polyhedron P'_j and P_j (lower left region) and between the right-shift polyhedron P''_j and P_j (upper right region). For each job $j \in J$, energetic reasoning takes effect only on such intervals.

that corresponds to a line segment $T_2^M(j)$ or $T_3(j)$ and $t_1 \geq e_j + p_j$ we delete the t_2 segment from the list because it is not defined according to Lemma 3 (lines 10–12). By construction of the line segments $T_1(j)$ and $T_1^M(j)$ it is ensured that all intersection points are traversed by the algorithm. In particular, insertions and deletions are always performed at the current list element. Therefore, all modifications to the original algorithm can be implemented in $\mathcal{O}(1)$. The number of iterated intervals remains $\mathcal{O}(n^2)$ in general, hence the complexity of our overload checking algorithm is also $\mathcal{O}(n^2)$.

Note that the intersection relations $\left(T_2(i) \cup T_2^M(i)\right) \cap T_3(i)$ of $\mathcal{T}'_{ij}$ in (15) are not included in our algorithms since they are symmetric to $\left(T_1(i) \cup T_1^M(i)\right) \cap T_3(i)$ at time $t = 0$. We execute our algorithm also for its symmetric version to include all relevant intervals.

5   Energetic Reasoning Propagation

In this section, we extend the overload checking algorithm of Sect. 4 by start and end time propagations based on energetic reasoning, as in (7).

Extension. In principle, finding a left-shift energetic reasoning propagation for a job $j \in J$ corresponds to finding an energy overload restricting to polyhedra P'_j and P_i for all $i \in J \setminus \{j\}$, where P'_j emerges from P_j by setting $l_j = e_j + p_j$. Analogously for right-shift propagations, where P''_j emerges from P_j by setting $e_j = l_j - p_j$, see Fig. 2. In order to include all relevant intervals that are implied by the polyhedra P'_j and P''_j we need to take the extended line segments

$$T_1(j) = \{(e_j, t_2) \in \mathbb{R}^2 \mid t_2 \geq t_1\}, \; T_1^M(j) = \{(l_j - p_j, t_2) \in \mathbb{R}^2 \mid l_j - p_j \leq t_2\}$$
$$T_2(j) = \{(t_1, l_j) \in \mathbb{R}^2 \mid t_1 \leq l_j\}, \; T_2^M(j) = \{(t_1, e_j + p_j) \in \mathbb{R}^2 \mid t_1 \leq e_j + p_j\}$$

of Lemma 3 for all jobs $j \in J$. Now these line segments corresponds to vertical and horizontal lines that cross the entire interval plane $t_1 \leq t_2$. Therefore, any

dynamic update of our overload checking algorithm of Sect. 4 becomes obsolete for these line segments. Hence, only the diagonal line segments of $T_3(j)$ for each $j \in J$ are dynamically updated. This gives an equivalent but simpler characterization of relevant intervals for energetic reasoning, as given in [3]. On the basis of this characterization, we also get a simple $\mathcal{O}(n^3)$ energetic reasoning propagation algorithm which is equivalent to [3], see Algorithm 2 in the appendix. In the following we consider the set $\mathcal{T}$ as defined in (15) but using the extended line segments.

Problem Decomposition. We call an energetic reasoning propagation algorithm *exact*, if it computes

$$\max_{(t_1,t_2)\in\mathcal{T}:\ \omega_j(t_1,t_2)>0} t_2 - \mu_j(t_1,t_2) + \left\lceil \frac{\omega(t_1,t_2)}{d_j} \right\rceil \tag{16}$$

for all jobs $j \in J$, that is the maximum earliest start time update for all jobs $j \in J$ according to (7). In particular, problem (16) implies two nested subproblems for each job $j \in J$. First, we have to find intervals $(t'_1, t'_2) \in \mathcal{T}$ of positive energy overload $\omega_j(t'_1, t'_2) > 0$. Second, among such intervals $(t'_1, t'_2) \in \mathcal{T}$ with $\omega_j(t'_1, t'_2) > 0$ we have to determine an interval $(t_1, t_2) \in \mathcal{T}$ that yields the maximum update with respect to (16). Our idea is to decompose (16) and to compute the maxima of the two incorporated functions

$$\max_{(t_1,t_2)\in\mathcal{T}} \omega_j(t_1, t_2) \tag{17}$$

$$\max_{(t_1,t_2)\in\mathcal{T}} t_2 - \mu_j(t_1, t_2) + \left\lceil \frac{\omega(t_1,t_2)}{d_j} \right\rceil \tag{18}$$

for all jobs $j \in J$. If for any job $j \in J$ the maxima of functions (17) or (18) are attained at an interval $(t_1, t_2) \in \mathcal{T}$ and it holds $\omega_j(t_1, t_2) > 0$ we update the earliest start time according to (7).

The approach of Bonifas [2] reversely computes $\max_{j\in J} \omega_j(t_1, t_2)$ for each relevant time interval $(t_1, t_2) \in \mathcal{T}$. If the time interval $(t_1, t_2) \in \mathcal{T}$ attains its maximum at job $j \in J$ he updates the earliest start time e_j according to (7). Since one job can dominate on many intervals, his algorithm generally takes $\mathcal{O}(n^3 \log n)$ to propagate all jobs. In the following we construct an $\mathcal{O}(n^2 \log n)$ algorithm that propagates all jobs according to (17) and (18). Due to the additional computation of (18) each propagation is equivalent or stronger than in [2]. Hence, our approach dominates Bonifas' algorithm with respect to complexity and propagation strength.

In particular, the computation of (17) yields at least one possible energetic reasoning propagation for every job. But in general, our approach may not detect the maximum propagations in one step. Thus, we apply our algorithm until a fixpoint is reached. To our knowledge it is unknown if the number of fixpoint iterations for energetic reasoning is polynomially bounded. At least, it is not strongly polynomial, see Mercier and van Hentenryck [14]. In practice, however, it rarely takes more than two iterations to reach the fixpoint and mostly the fixpoints of our approach and exact energetic reasoning are equal.

Geometry. The overload checking algorithm of Sect. 4 first loops over all t_1 values and then over all t_2 values with $(t_1, t_2) \in \mathcal{T}$ while checking for potential energy overloads $\omega(t_1, t_2) > 0$. In the following we consider a fixed iteration of the main t_1-loop for a fixed value $\bar{t}_1$.

According to fixed $\bar{t}_1$, we reformulate functions (17) and (18) equivalently as

$$\max_{(\bar{t}_1, t_2) \in \mathcal{T}} \omega(\bar{t}_1, t_2) + d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2)) \tag{19}$$

$$\max_{(\bar{t}_1, t_2) \in \mathcal{T}} \omega(\bar{t}_1, t_2) + d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2)) \tag{20}$$

for all jobs $j \in J$ by using definition (6) and the fact that we only need the intervals $(\bar{t}_1, t_2) \in \mathcal{T}$ where the maximum is attained. Now, the values $\omega(\bar{t}_1, t_2)$ with $(\bar{t}_1, t_2) \in \mathcal{T}$ form the set of points $\mathcal{P} = \{(t_2, \omega(\bar{t}_1, t_2) \in \mathbb{R}^2 \mid (\bar{t}_1, t_2) \in \mathcal{T}\}$ which are collected during the overload checking algorithm. The remaining two piecewise linear functions $d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ and $d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ will be decomposed into a set of line segments $\mathcal{L}_j$ in $\mathbb{R}^2$, see Lemmas 4 and 5.

Geometrically in $\mathbb{R}^2$, problems (19) and (20) compute for each job $j \in J$ and each line segment $l \in \mathcal{L}_j$ the point $(t_2, \omega(\bar{t}_1, t_2)) \in \mathcal{P}$ in the domain of line l such that the sum of their function values at t_2 is maximal. Since by Lemmas 4 and 5 it holds $|\mathcal{L}_j| \in \mathcal{O}(1)$, we select for each job $j \in J$ the maximum value among all line segments $l \in \mathcal{L}_j$. In Sect. 6 we introduce a new sweep line algorithm that solves this problem in $\mathcal{O}((|\mathcal{L}| + |\mathcal{P}|) \cdot \log(|\mathcal{L}| + |\mathcal{P}|))$, where $\mathcal{L} = \bigcup_{j \in J} \mathcal{L}_j$. From $|\mathcal{L}| \in \mathcal{O}(n)$ and $|\mathcal{P}| \in \mathcal{O}(n)$ it follows that the subproblems (19) and (20) can be solved in $\mathcal{O}(n \log n)$, which gives a total running time of $\mathcal{O}(n^2 \log n)$.

Line Segment Decomposition. Lemmas 4 and 5 show how the functions $d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ and $d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ can be decomposed into line segments $\mathcal{L}_j$ for all jobs $j \in J$. This applies only to the left-shift case. For the right-shift case, Lemmas 6 and 7 provide line segment decompositions for the analogous functions $d_j \cdot (\mu_j^{right}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ and $-d_j \cdot (\bar{t}_1 + \mu_j(\bar{t}_1, t_2))$. The simultaneous consideration of left- and right-shift propagations enables us to compute all interesting propagations in one step, compare Sect. 7 and Algorithm 4. We restrict both cases to their specific interval region where energetic reasoning propagations may occur, see Fig. 2. For this, define $\theta_1 = \max\{e_j, \bar{t}_1\}$, $\theta_2 = \min\{e_j + p_j, l_j - p_j\}$, $\theta_3 = \max\{e_j + p_j, l_j - p_j\}$ and $\theta_4 = \min\{l_j, l_j + e_j - \bar{t}_1\}$.

Lemma 4. *For any job $j \in J$ and $\bar{t}_1 \leq \theta_2$ the piecewise linear function $f_j(t_2) = d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta_1, \theta_4]$ decomposes into*

$$\begin{aligned} f_j^1(t_2) &= d_j \cdot (t_2 - \theta_1), & t_2 &\in [\theta_1, \theta_2] \\ f_j^2(t_2) &= d_j \cdot (\theta_2 - \theta_1), & t_2 &\in [\theta_2, \theta_3] \\ f_j^3(t_2) &= -d_j \cdot (t_2 - \theta_4), & t_2 &\in [\theta_3, \theta_4]. \end{aligned}$$

Lemma 5. *For any job $j \in J$ and $\bar{t}_1 \leq \theta_2$ the piecewise linear function $f_j(t_2) = d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta_1, \theta_4]$ decomposes into the linear functions*

$$f_j^1(t_2) = d_j \cdot t_2 , \quad t_2 \in [\theta_1, l_j - p_j]$$
$$f_j^2(t_2) = d_j \cdot (l_j - p_j) , \quad t_2 \in [l_j - p_j, \theta_4].$$

Lemma 6. *Let $\theta' = \max\{\theta_3, \theta_4, \bar{t}_1\}$. For any job $j \in J$ and $\bar{t}_1 \in [e_j, l_j]$ the piecewise linear function $f_j(t_2) = d_j \cdot (\mu_j^{right}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta', \infty)$ decomposes into the linear function segments*

$$f_j^1(t_2) = d_j \cdot (t_2 - \theta') , \quad t_2 \in [\theta', l_j]$$
$$f_j^2(t_2) = d_j \cdot (l_j - \theta') , \quad t_2 \in [l_j, \infty).$$

Lemma 7. *Let $\theta' = \max\{\theta_3, \theta_4, \bar{t}_1\}$. For any job $j \in J$ and $\bar{t}_1 \in [e_j, l_j]$ the piecewise linear function $f_j(t_2) = -d_j \cdot (\bar{t}_1 + \mu_j(\bar{t}_1, t_2))$ is constant on $[\theta', \infty)$.*

6   Sweep Line Algorithm

In this section we introduce a new sweep line algorithm that solves the geometric problems (19) and (20) that occur during energetic reasoning. Compared to Sect. 5, we restate the problem more generally.

Let $\mathcal{P}$ be a set of pairwise distinct points in $\mathbb{R}^2$ where each point $q \in \mathcal{P}$ has coordinates $(x_q, y_q) \in \mathbb{R}^2$. Additionally, let $\mathcal{L}$ be a set of line segments in $\mathbb{R}^2$ where each line segment $l \in \mathcal{L}$ is given a slope $a_l \in \mathbb{R}$, an intercept $b_l \in \mathbb{R}$ and an interval $[\underline{x}_l, \overline{x}_l] \subset \mathbb{R}$. Thus, each line segment $l \in \mathcal{L}$ corresponds to the set of points $(x, y) \in \mathbb{R}^2$ that satisfy $y = a_l \cdot x + b_l$ with $x \in [\underline{x}_l, \overline{x}_l]$. For each line segment $l \in \mathcal{L}$ we want to compute a point $q \in \mathcal{P}$ with $x_q \in [\underline{x}_l, \overline{x}_l]$ that has maximum y-distance to the line segment l. More formally, we compute

$$\max_{q \in \mathcal{P}: x_q \in [\underline{x}_l, \overline{x}_l]} a_l \cdot x_q + y_q + b_l \tag{21}$$

for all line segments $l \in \mathcal{L}$. Since b_l is constant in this term we reduce to

$$\max_{q \in \mathcal{P}: x_q \in [\underline{x}_l, \overline{x}_l]} a_l \cdot x_q + y_q \tag{22}$$

for all line segments $l \in \mathcal{L}$. If there is no $q \in \mathcal{P}$ with $x_q \in [\underline{x}_l, \overline{x}_l]$ we assume the function has value $-\infty$. Our approach is to dualize problem (22) and translate it from the (x, y)-plane to the (a, y)-plane with respect to the slopes a_l of the line segments $l \in \mathcal{L}$, see Fig. 3. In this dual setting, each point $q \in \mathcal{P}$ corresponds to a line with slope x_q and intercept y_q that contains all points $(a, y) \in \mathbb{R}^2$ with $y = x_q \cdot a + y_q$. Moreover, each line segment $l \in \mathcal{L}$ translates to a point $(a_l, 0) \in \mathbb{R}^2$. The equivalent dual problem is to compute for each point $(a_l, 0)$ with $l \in \mathcal{L}$ the line $q \in \mathcal{P}$ with $x_q \in [\underline{x}_l, \overline{x}_l]$ of maximum value $y = x_q \cdot a_l + y_q$. The difficulty of the dual problem is to consider for each point $(a_l, 0) \in \mathbb{R}^2$ with
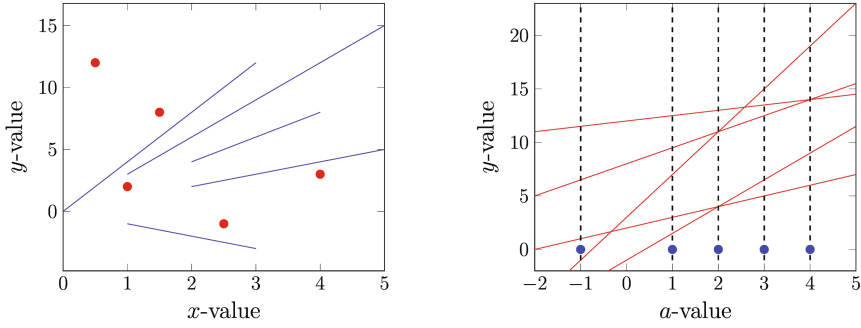
Fig. 3. Primal problem (left): given a set of points $(x_q, y_q) \in \mathcal{P}$ and a set of line segments $l \in \mathcal{L}$ with slope a_l. Dual problem (right): converts to a set of lines $q \in \mathcal{P}$ with slope x_q, intercept y_q and a set of evaluation points $(a_l, 0)$ for all $l \in \mathcal{L}$ where the sweep line (dashed) is evaluated. The interval tree B stores the state of the sweep line.

$l \in \mathcal{L}$ a subset of lines $q \in \mathcal{P}$ with slopes x_q in the range $[\underline{x}_l, \overline{x}_l]$. Therefore, an upper envelope computation of all dual lines $q \in \mathcal{P}$ is not sufficient.

In the following we will stick to the *dual setting*, that means we consider $\mathcal{P}$ as a set of lines and $\mathcal{L}$ as a set of points in $\mathbb{R}^2$. A detailed pseudo-code of the following algorithms can be found in Algorithms 5–8 in the appendix.

Sweeping (Algorithm 5). The sweep line algorithm sweeps over all points $(a_l, 0) \in \mathbb{R}^2$ with $l \in \mathcal{L}$ in non-decreasing order of a_l. All a_l values are stored as *evaluation events* in a *min-heap* H. At each point $(a_l, 0)$ we evaluate function (22), which can be done efficiently for any interval $[\underline{x}_l, \overline{x}_l]$ by using a *binary interval tree* B which stores the current state of the sweep line. While sweeping over all a_l values with $l \in \mathcal{L}$ the state of the sweep line changes, so the interval tree B must be updated dynamically. Therefore, the heap H additionally stores *resolve events* which constitute events where the tree structure must be updated. New resolve events are added dynamically during the sweep.

The main sweep line algorithm successively extracts the minimum element from the heap H. If it is an evaluation event, we call the subroutine *evaluate* and if it is a resolve event we call the subroutine *resolve*. The main concepts of the sweep line algorithm are explained in the following.

Interval Tree (Algorithm 6). Let V_B denote the set of nodes of the interval tree B. Each tree node $v \in V_B$ stores four data members: an *interval* $[\underline{x}_v, \overline{x}_v] \subset \mathbb{R}$, a *dominating line* $\pi_v \in \mathcal{P}$, a *resolve point* $\alpha_v \in \mathbb{R}$ and a *minimum resolve point* $\beta_v \in \mathbb{R}$. The data members α_v, β_v and π_v change dynamically while sweeping over all a_l values with $l \in \mathcal{L}$. For an initial sweep value $a_0 \in \mathbb{R}$ the tree is build up recursively from bottom to top. The leaves of B, from left to right, correspond to lines $q \in \mathcal{P}$ sorted by x_q first and by y_q second in non-decreasing order. The data members of a *leaf node* $v \in V_B$ that is associated with one line $q \in \mathcal{P}$ is initialized by $[\underline{x}_v, \overline{x}_v] = [x_q, x_q]$, $\pi_v = q$, $\alpha_v = \infty$ and $\beta_v = \infty$.

Conversely, the data members of a *non-leaf node* $v \in V_B$ with child nodes $v.left \in V_B$ and $v.right \in V_B$ are defined recursively as follows. The interval $[\underline{x}_v, \overline{x}_v] = [\underline{x}_{v.left}, \overline{x}_{v.right}]$ spans the intervals of the child nodes of v. Its dominating line π_v is equal to the line $q \in \{\pi_{v.left}, \pi_{v.right}\}$ that has the higher value of $a_0 \cdot x_q + y_q$ (lines 15–16). Furthermore, for the dominating lines $q = \pi_{v.left}$ and $q' = \pi_{v.right}$ the resolve point $\alpha_v = \frac{y_q - y_{q'}}{x_{q'} - x_q}$ equals the intersection point of the lines q and q', if $x_q \neq x_{q'}$. Otherwise, if $x_q = x_{q'}$ set $\alpha_v = \infty$ (lines 18–24). Finally, let $\beta_v = \min\{\alpha_{v.left}, \alpha_{v.right}, \beta_{v.left}, \beta_{v.right}\}$ be the minimum value of a resolve point of any node in the subtree rooted at v (line 14). If, during the construction, it holds $\alpha_v < \beta_v$ for some $v \in V_B$ we add a resolve event with value α_v to the heap H (lines 22–23), see also *resolve*.

Evaluate (Algorithm 7). This subroutine evaluates (22) for a sweep value a_l and an interval $[\underline{x}_l, \overline{x}_l]$ with $l \in \mathcal{L}$. For this, we descend the interval tree B recursively from the root along nodes $v \in V_B$ with $[\underline{x}_v, \overline{x}_v] \cap [\underline{x}_l, \overline{x}_l] \neq \emptyset$. For nodes $v \in V_B$ with $[\underline{x}_v \overline{x}_v] \subseteq [\underline{x}_l, \overline{x}_l]$ function (22) can be evaluated in $\mathcal{O}(1)$ because the dominating line $\pi_v \in \mathcal{P}$ (line 11) yields the maximum value of (22) in the interval $[\underline{x}_v, \overline{x}_v]$ by construction. Therefore, the recursion descends the tree B only along the interval limits $\underline{x}_l$ and $\overline{x}_l$. Hence, one evaluation takes $\mathcal{O}(\log |\mathcal{P}|)$.

Resolve (Algorithm 8). This subroutine resolves a node $v \in V_B$ at its sweep value α_v. Recall that α_v denotes the intersection value of the dominating lines $\pi_{v.left}, \pi_{v.right} \in \mathcal{P}$. Since resolving means that $\pi_{v.right}$ replaces $\pi_{v.left}$ as dominating line with respect to (22) for all sweep values $a_l > \alpha_v$ we set $\pi_v = \pi_{v.right}$. Additionally, we set $\alpha_v = \infty$ since the lines $\pi_{v.left}$ and $\pi_{v.right}$ have no future intersection because $\pi_{v.right}$ has the higher slope by construction (lines 1–3).

By its recursive definition, changing the dominating line $\pi_v \in \mathcal{P}$ may change the values of α_u, β_u and π_u of all nodes u on the path from v to the root of B. Thus, we propagate those values along this path (lines 4–18). In particular, α_u is updated only if it was not already resolved, that is $\alpha_u < \infty$. If, after the propagation, it holds $\alpha_u < \beta_u$ for any node u on the path from v to the root of B then the resolve event at α_u is added to the heap H (line 14). This is because we only add the resolve event at the intersection point α_v to the heap H, if α_v does not change due to recursion. Otherwise, if $\beta_v \leq \alpha_v$ then resolving a child node of v may still affect the recursive value of α_v. In this case, we add at most one resolve event for each tree node which is crucial for the running time of our algorithm. Since only data members along the path from a node $v \in V_B$ to the root of B are changed, one resolve takes $\mathcal{O}(\log |\mathcal{P}|)$.

Theorem 3. *The sweep line algorithm runs in $\mathcal{O}((|\mathcal{L}| + |\mathcal{P}|) \cdot \log(|\mathcal{L}| + |\mathcal{P}|))$.*

Proof. The interval tree is initialized in $\mathcal{O}(|\mathcal{P}|)$. The event heap H contains at most $\mathcal{O}(|\mathcal{P}|)$ resolve events and $\mathcal{O}(|\mathcal{L}|)$ evaluation events, so extracting all elements gives $\mathcal{O}((|\mathcal{L}| + |\mathcal{P}|) \cdot \log(|\mathcal{L}| + |\mathcal{P}|))$. The main loop extracts $\mathcal{O}(|\mathcal{L}| + |\mathcal{P}|)$ events from the heap H and each event is either evaluated or resolved, where each has complexity $\mathcal{O}(\log |\mathcal{P}|)$. Hence, the statement follows. $\square$

7   Exact Intervals

Finally, we characterize interval subsets $\mathcal{T}_j \subseteq \mathcal{T}$ for all jobs $j \in J$ where our sweep line propagation algorithm of Sect. 5 is exact. That means, if the interval $(t_1, t_2) \in \mathcal{T}$ yields the maximum earliest start time update (7) for job $j \in J$ and it holds $(t_1, t_2) \in \mathcal{T}_j$ then the sweep line propagator finds this propagation. Therefore, recall the functions

$$\max_{(\bar{t}_1, t_2) \in \mathcal{T}} \omega(\bar{t}_1, t_2) + d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2)) \tag{23}$$

$$\max_{(\bar{t}_1, t_2) \in \mathcal{T}} \omega(\bar{t}_1, t_2) + d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2)) \tag{24}$$

as given in (19) and (20) for fixed value $\bar{t}_1 \in \mathbb{R}$.

Lemma 8. *If the slopes of the functions (23) and (24) coincide on an interval $[\underline{t}_2, \bar{t}_2] \subset \mathbb{R}$ then both functions on the interval $[\underline{t}_2, \bar{t}_2]$ attain their maximum at the same point $(\bar{t}_1, t_2) \in \mathcal{T}$ with $t_2 \in [\underline{t}_2, \bar{t}_2]$, if the maximum exists.*

Lemma 8 implies that the two nested subproblems (23) and (24) of the exactness condition (16) attain their maximum at the same point $t_2 \in [\underline{t}_2, \bar{t}_2]$, if their slopes are equal. Thus, only one function of (23) and (24) must be considered. Since the sweep line algorithm of Sect. 5 computes such maxima, we obtain exact energetic reasoning propagations on the interval $[\underline{t}_2, \bar{t}_2]$. This suggests to characterize intervals where the difference $d_j \cdot (t_2 - \mu_j^{left}(\bar{t}_1, t_2))$ of functions (23) and (24) is constant. Analogously for the right-shift case, we study intervals where the function $\mu_j^{right}(\bar{t}_1, t_2) - 2 \cdot \mu_j(\bar{t}_1, t_2)$ is constant.

Recall Fig. 2 and consider the following subdivision of interval areas

$$\mathcal{T}_j^A = \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 \in [e_j, l_j],\ t_2 \in [l_j, \infty)\}$$
$$\mathcal{T}_j^B = \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 \in [e_j, \theta_2],\ t_2 \in [e_j + p_j, \theta_4]\}$$
$$\mathcal{T}_j^{B'} = \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 \in [e_j, l_j - p_j],\ t_2 \in [\max\{\theta_3, \theta_4\}, l_j]\}$$
$$\mathcal{T}_j^C = \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 \in [e_j, \theta_2],\ t_2 \in [t_1, e_j + p_j]\}$$

where $\theta_1 = \max\{e_j, \bar{t}_1\}$, $\theta_2 = \min\{e_j + p_j, l_j - p_j\}$, $\theta_3 = \max\{e_j + p_j, l_j - p_j\}$ and $\theta_4 = \min\{l_j, l_j + e_j - \bar{t}_1\}$.

Lemma 9. *For fixed value $\bar{t}_1 \in [e_j, l_j]$ the function $\mu_j^{right}(\bar{t}_1, t_2) - 2 \cdot \mu_j(\bar{t}_1, t_2)$ has slope zero for all $t_2 \in [l_j, \infty)$.*

Lemma 10. *For fixed value $\bar{t}_1 \in [e_j, \theta_2]$ the function $t_2 - \mu_j^{left}(\bar{t}_1, t_2)$ has slope one the interval $t_2 \in [e_j + p_j, \theta_4]$.*

Lemma 11. *For fixed value $\bar{t}_1 \in [e_j, \theta_2]$ the function $\mu_j^{right}(\bar{t}_1, t_2) - 2 \cdot \mu_j(\bar{t}_1, t_2)$ has slope one in the interval $[\max\{\theta_3, \theta_4\}, l_j]$.*

Lemma 12. *For fixed value $\bar{t}_1 \in [e_j, \theta_2]$ the function $t_2 - \mu_j^{left}(\bar{t}_1, t_2)$ has slope zero in the interval $[t_1, \theta_2]$.*

From Lemmas 9 and 12 we deduce that our approach is exact on the interval sets $\mathcal{T}_j^A$ and $\mathcal{T}_j^C$. Thus, we execute our algorithm also for the symmetric version of the problem to imply exactness on their symmetric copies $\mathcal{T}_j^{A'}$ and $\mathcal{T}_j^{C'}$, see Fig. 2. In contrast, Lemmas 10 and 11 show that our approach is not exact on the symmetric interval sets $\mathcal{T}_j^B$ and $\mathcal{T}_j^{B'}$. But in practice, both sets form only a minor part of the whole interval region where energetic reasoning is propagated. Fr our computational results, we implemented two versions. One version omits the line segments that belong to the non-exact interval regions and the other includes the slopes of both functions (23) and (24) to enhance our chance to find the exact propagation, see Algorithm 4.

8   Computational Results

Our algorithms are implemented in C++ using Linux GCC compiler version 4.8.4 on a 3.20 GHz Intel Xeon CPU and 16GB RAM. The test set is taken from RCPSP instances of the PSPLIB [16] and contains 480 instances of 30 jobs, 480 instances of 60 jobs and 600 instances of 120 jobs.

We implemented the basic overload checker of Derrien et al. [3] (CD) and our improved dynamic overload checker (CC) of Sect. 4. As energetic reasoning propagators, we implemented the original $\mathcal{O}(n^3)$ propagator of Baptiste et al. [5] (ERB) and a simpler but equivalent version of the $\mathcal{O}(n^3)$ propagator of Derrien et al. [3] (ERD), see Algorithm 2. Moreover, we implemented two versions of our $\mathcal{O}(n^2 \log n)$ sweep line propagator. The full version (SWF) adds all line segments of the exact and non-exact interval regions, compare Algorithm 4 and Sect. 7. For the non-exact regions, only lines with the slopes of the corresponding overload and update function are added. The relaxed version (SWR) considers only line segments of the exact interval regions. In addition to all algorithms, we apply a fast time-tabling and precedence propagator [9].

We compute lower bounds by destructive improvement [8]. In order to show the real performance of the algorithms we first apply the static *SetTimes* [17] branching rule. To show the practical performance we also apply a dynamic branching rule, similar to Schutt et al. [9], which stores a score value for every job that is increased by one, if fixing this job to its earliest start time leads to a direct failure, otherwise the score is decreased by one. We select the job with the highest score value and break ties with the earliest start and latest completion time. The time limit for each lower bound is 3600 s.

Tables 1 and 2 compare the results of the static and the dynamic branching rule. The column *opt* shows the number of optimally solved instances, ΔLB is the sum of the computed lower bounds normed to the weakest algorithm and *nodes/s* is the average number of nodes per second of the optimally solved instances. Our results show that the precise polyhedral interpretation of the dynamic overload checker (CC) leads to a speedup of factor two compared to the checker of Derrien et al. [3]. Due to this gain, the checker performs also very well as standalone algorithm combined with dynamic branching. We have to

Table 1. Results for the checkers and propagators using static branching.

	J30			J60			J120		
	opt	Δ LB	nodes/s	opt	Δ LB	nodes/s	opt	Δ LB	nodes/s
CD	375	90	815.16	329	0	253.84	159	0	78.86
CC	382	150	1577.41	330	31	590.97	159	25	198.42
ERB	376	0	36.49	343	31	6.38	167	100	1.27
ERD	386	87	137.14	344	62	21.05	168	128	4.20
SWF	386	52	288.04	343	48	84.01	169	97	29.09
SWR	391	97	456.73	342	80	139.65	169	132	50.85

Table 2. Results for the checkers and propagators using dynamic branching.

	J30			J60			J120		
	opt	Δ LB	nodes/s	opt	Δ LB	nodes/s	opt	Δ LB	nodes/s
CD	458	48	1065.78	386	168	266.79	211	93	90.21
CC	461	78	1747.14	389	210	596.91	217	196	205.78
ERB	446	0	40.93	369	0	6.58	184	0	1.38
ERD	454	44	137.25	380	71	22.01	197	61	4.70
SWF	455	27	299.33	377	83	80.26	209	136	30.29
SWR	460	52	483.42	387	151	134.84	214	205	53.05

note that the PSPLIB instances are rather cumulative than disjunctive, that is pure checkers perform very reasonable compared to pure energetic approaches.

Our sweep line propagators also show a very positive performance. In terms of computation time, full and relaxed sweep line propagation highly dominate the previous energetic reasoning propagators by a factor up to twelve on large instances. In terms of propagation power, however, full sweep line propagation (SWF) is slightly inferior to the propagator of Derrien et al. [3] using static branching. Using dynamic branching, full sweep line propagation dominates again. In particular, relaxing the non-exact line segments (SWR) results in a vast computational speedup where much more propagations are performed in the same time. The relationship between almost the same propagation power as exact energetic reasoning and much faster computation time lets the relaxed sweep line algorithm (SWR) outperform all other energetic reasoning algorithms.

On the 120 job test set, the relaxed sweep line algorithm (SWR) further improved four best known lower bounds: 103 (34_2), 128 (47_6), 104 (59_5), 88 (60_3) and our dynamic checker (CC) improved six additional best known lower bounds: 116 (12_6), 218 (36_3), 119 (47_3), 128 (47_10), 124 (53_10), 126 (58_9). The improvement is +1 for each instance, we compared with [18].

9   Conclusion

In this paper we propose an improved overload checking algorithm and a new energetic reasoning propagation algorithm for solving the CuSP. Our approaches are derived from a novel polyhedral interpretation of energetic reasoning. On that basis, we develop practically efficient algorithms that improve the current state of the art. Further research may focus on practical refinements of the presented algorithms and the development of more sophisticated combinatorial methods from a related polyhedral background.

A   Proofs

A.1   Proof of Lemma 2

We first show the following helping lemma.

Lemma 13. *Let $S \subseteq J$ be a job subset with $S \neq \emptyset$ and $P_S \neq \emptyset$. In addition, let $(t_1, t_2, \tilde{\mu}) \in P_S$ be a vertex of P_S with $\tilde{\mu}_j > 0$ for all $j \in S$. Then either one of the following holds:*

(i) $(t_1, t_2, \tilde{\mu})$ satisfies three inequalities of (8)–(11) with equality and all of them correspond to one job $j \in S$

(ii) $(t_1, t_2, \tilde{\mu})$ satisfies four inequalities of (8)–(11) with equality where two correspond to one job $i \in S$ and two correspond to one job $j \in S$ with $i \neq j$.

Proof. We first show that P_S has full dimension. Let $m = |S|$ and let $\delta_j \in \mathbb{R}^m$ be the j-th unit vector. Furthermore, let $0_m, 1_m \in \{0,1\}^m$ be the vectors that contain m zeros or one respectively. Consider the $m+2$ vectors $(-T, T, p_j \cdot \delta_j)_{j \in S}$, $(0, T, 0_m)$ and $(T, T, 0_m)$ where T denotes a large constant. We verify that these vectors are linearly independent and satisfy inequalities (8)–(13). Consequently, P_S contains $m + 2$ linearly independent vectors, so it has full dimension $m + 2$.

It follows that the vertex $(t_1, t_2, \tilde{\mu}) \in P_S$ satisfies $m+2$ inequalities of (8)–(13) with equality. If it satisfies inequality (12) or (13) with equality it holds $\tilde{\mu}_j = 0$ for some $j \in S$ which contradicts the assumption. Hence, we can restrict to inequalities (8)–(11) which yields the reduced constraint matrix $A \in \{0,1\}^{4 \cdot m \times m+2}$ of the form

$$A = \begin{pmatrix} 0_m & 0_m & I_m \\ 1_m & -1_m & I_m \\ 1_m & 0_m & I_m \\ 0_m & -1_m & I_m \end{pmatrix} \quad \begin{matrix} (8) \\ (9) \\ (10) \\ (11) \end{matrix}$$

where the first two columns of A correspond to variables t_1, t_2 and the last m columns correspond to variables $\tilde{\mu}_j$ with $j \in S$. Here, $I_m \in \{0,1\}^{m \times m}$ equals the $m \times m$ identity matrix.

Thus, the vertex $(t_1, t_2, \tilde{\mu}) \in P_S$ corresponds to a selection of $m + 2$ linearly independent rows of A whose associated submatrix we denote by A_B. Since every

column of A_B must contain at least one non-zero entry and each row of A has exactly one non-zero coefficient for some variable $\tilde{\mu}_j$ it follows that A_B contains m rows with non-zero entries for each variable $\tilde{\mu}_j$ with $j \in S$. The remaining two rows of A_B either have a non-zero entry for one job $j \in S$ or for two distinct jobs $i, j \in S$. This is equivalent to cases (i) and (ii) which proves the lemma. $\square$

Lemma 2. *Let $S \subseteq J$ be a job subset with $S \neq \emptyset$ and $P_S \neq \emptyset$. In addition, let $(t_1, t_2, \tilde{\mu}) \in P_S$ be a vertex of P_S with $\tilde{\mu}_j > 0$ for all $j \in S$. Then either one of the following holds:*

(i) *There is a job $j \in S$ such that $(t_1, t_2, \tilde{\mu}_j)$ is a vertex of P_j.*
(ii) *There are two distinct jobs $i, j \in S$ such that $(t_1, t_2, \tilde{\mu}_i, \tilde{\mu}_j)$ is the intersection of one edge of P_i and one edge of P_j and thus a vertex of $P_{i,j}$.*

Proof. It either holds case (i) or (ii) of Lemma 13 because the assumptions are equal. From the proof of Lemma 13, the polyhedra P_i and $P_{i,j}$ have dimensions three and four respectively. Case (i) of Lemma 13 implies that the projected vertex $(t_1, t_2, \tilde{\mu}_j)$ is a vertex of P_j.

An edge of P_j satisfies two inequalities of (8)–(11) with equality that correspond to job j. Therefore, case (ii) of Lemma 13 yields that the projected vertex $(t_1, t_2, \tilde{\mu}_i, \tilde{\mu}_j)$ is the intersection of one edge of P_i and one edge of P_j and hence a vertex of $P_{i,j}$. $\square$

A.2   Proof of Lemma 3

Lemma 3. *Given a job $j \in J$ and assume the projection of the polyhedron P_j to the (t_1, t_2)-plane. The projected line segments of the edges of P_j that contain a vertex $(t_1, t_2, \tilde{\mu}_j)$ of P_j with $\tilde{\mu}_j > 0$ are given by*

$$T_1(j) = \{(e_j, t_2) \in \mathbb{R}^2 \mid t_2 \geq l_j\}$$
$$T_2(j) = \{(t_1, l_j) \in \mathbb{R}^2 \mid t_1 \leq e_j\}$$
$$T_3(j) = \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 + t_2 = e_j + l_j, \ e_j \leq t_1 \leq \min\{e_j + p_j, l_j - p_j\}\}$$
$$T_1^M(j) = \{(l_j - p_j, t_2) \in \mathbb{R}^2 \mid l_j - p_j \leq t_2 \leq e_j + p_j\}$$
$$T_2^M(j) = \{(t_1, e_j + p_j) \in \mathbb{R}^2 \mid l_j - p_j \leq t_1 \leq e_j + p_j\}.$$

Proof. By the proof of Lemma 13, it suffices to restrict to inequalities (8)–(11). An edge of P_j satisfies two inequalities of (8)–(11) with equality. Thus, there are six possible cases:

(i) If inequalities (8) and (10) hold with equality then it holds $\tilde{\mu}_j = p_j = e_j + p_j - t_1$ which implies $t_1 = e_j$. By inequalities (11) and (9) it follows $t_2 \geq l_j$ and $t_2 \geq e_j + p_j$.
(ii) If inequalities (8) and (11) hold with equality then it holds $\tilde{\mu}_j = p_j = t_2 - l_j + p_j$ which implies $t_2 = l_j$. By inequalities (10) and (9) it follows $t_1 \leq e_j$ and $t_1 \leq l_j - p_j$.

(iii) If inequalities (10) and (11) hold with equality then it holds $\tilde{\mu}_j = e_j + p_j - t_1 = t_2 - l_j + p_j$ which implies $t_1 + t_2 = e_j + l_j$. By inequalities (8) and (9) it follows $t_1 \geq e_j$ and $t_1 \leq l_j - p_j$. In addition, inequality (13) yields $t_1 \leq e_j + p_j$.

(iv) If inequalities (9) and (11) hold with equality then it holds $\tilde{\mu}_j = t_2 - t_1 = t_2 - l_j + p_j$ which implies $t_1 = l_j - p_j$. By inequalities (10) and (8) it follows $t_2 \leq e_j + p_j$ and $t_2 \leq l_j$. In addition, inequality (12) yields $t_2 \geq l_j - p_j$.

(v) If inequalities (9) and (10) hold with equality then it holds $\tilde{\mu}_j = t_2 - t_1 = e_j + p_j - t_1$ which implies $t_2 = e_j + p_j$. By inequalities (11) and (8) it follows $t_1 \geq l_j - p_j$ and $t_1 \geq e_j$. In addition, inequality (12) yields $t_1 \leq e_j + p_j$.

(vi) If inequalities (8) and (9) hold with equality then it holds $\tilde{\mu}_j = p_j = t_2 - t_1$. By inequalities (10) and (11) it follows $t_1 \leq e_j$ and $t_2 \geq l_j$. Adding both yields $l_j - e_j \leq t_2 - t_1 = p_j \leq l_j - e_j$ which implies $p_j = l_j - e_j$. Thus, it holds $t_1 = e_j$ and $t_2 = l_j$. Therefore, all inequalities of (8)–(11) are satisfied with equality. This case is already included in cases (i)–(v).

Since $e_j \leq l_j - p_j$ and $e_j + p_j \leq l_j$ always holds the cases (i)-(v), in order of appearance, correspond to the line segments $T_1(j), T_2(j), T_3(j), T_1^M(j), T_2^M(j)$ which proves the lemma. □

A.3   Proof of Theorem 2

Theorem 2. *If $(t_1, t_2) \in \mathbb{R}^2$ is a time interval of maximum energy overload then it holds $(t_1, t_2) \in \mathcal{T}$.*

Proof. By Lemma 1 there exists a job subset $S \subseteq J$ with $S \neq \emptyset$ such that $(t_1, t_2, \tilde{\mu}) \in \mathbb{R}^{|S|+2}$ is a vertex of P_S with $\tilde{\mu}_j > 0$ for all $j \in S$. Therefore, Lemma 2 holds. We distinguish between cases (i) and (ii) of Lemma 2.

If case (i) holds then there is a job $j \in S$ such that $(t_1, t_2, \tilde{\mu}_j)$ is a vertex of P_j. Since P_j is a three-dimensional polyhedron the vertex $(t_1, t_2, \tilde{\mu}_j)$ of P_j has at least three incident edges. By Lemma 3, the only intersection points of at least three projected edges of P_j are $(t_1, t_2) = (e_j, l_j)$ and $(t_1, t_2) = (l_j - p_j, e_j + p_j)$, if $j \in J^M$. This case is equivalent to $(t_1, t_2) \in \mathcal{T}_j$ and $(t_1, t_2) \in \mathcal{T}_j^M$, if $j \in J^M$.

Otherwise, if case (ii) holds then there are two distinct jobs $i, j \in S$ such that (t_1, t_2) is an intersection point of the projected edges of P_i and P_j respectively. Since $T_1(i), T_1^M(i)$ are vertical, $T_2(i), T_2^M(i)$ horizontal and $T_3(i)$ diagonal line segments the possible intersection relations are vertical-horizontal, vertical-diagonal and horizontal-diagonal. The relation vertical-horizontal corresponds to $(t_1, t_2) \in \mathcal{T}_{ij}$ and the relations vertical-diagonal and horizontal-diagonal to $(t_1, t_2) \in \mathcal{T}_{ij}'$. If the line segments of jobs i and j intersect in more than one point, we can always find an intersection point of the previous characterizations along the intersecting line. It follows that $(t_1, t_2) \in \mathcal{T}$ which shows the theorem. □

A.4   Proof of Lemmas 4–7

Lemma 4. *For any job $j \in J$ and $\bar{t}_1 \leq \theta_2$ the piecewise linear function $f_j(t_2) = d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta_1, \theta_4]$ decomposes into the linear function segments*

$$f_j^1(t_2) = d_j \cdot (t_2 - \theta_1), \quad t_2 \in [\theta_1, \theta_2]$$
$$f_j^2(t_2) = d_j \cdot (\theta_2 - \theta_1), \quad t_2 \in [\theta_2, \theta_3]$$
$$f_j^3(t_2) = -d_j \cdot (t_2 - \theta_4), \quad t_2 \in [\theta_3, \theta_4].$$

Proof. Since $\bar{t}_1 \leq \theta_2$, the function $\mu_j^{left}(\bar{t}_1, t_2)$ has slope one in the interval $t_2 \in [\theta_1, e_j + p_j]$ and zero otherwise. The function $\mu_j(\bar{t}_1, t_2)$ has slope one in the interval $t_2 \in [l_j - p_j, \theta_4]$ and zero otherwise. Hence $\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2)$ has slope one the interval $[\theta_1, \theta_2]$, constant slope in $[\theta_2, \theta_3]$ and slope minus one in $[\theta_3, \theta_4]$. Scaling by d_j shows the statement. $\square$

Lemma 5. *For any job $j \in J$ and $\bar{t}_1 \leq \theta_2$ the piecewise linear function $f_j(t_2) = d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta_1, \theta_4]$ decomposes into the linear function segments*

$$f_j^1(t_2) = d_j \cdot t_2, \quad t_2 \in [\theta_1, l_j - p_j]$$
$$f_j^2(t_2) = d_j \cdot (l_j - p_j), \quad t_2 \in [l_j - p_j, \theta_4].$$

Proof. Since $\bar{t}_1 \leq \theta_2$, the function $\mu_j(\bar{t}_1, t_2)$ has slope zero in $[\theta_1, l_j - p_j]$ and slope one in the interval $[l_j - p_j, \theta_4]$. Hence, the function $t_2 - \mu_j(\bar{t}_1, t_2)$ has slope one in the interval $[\theta_1, l_j - p_j]$ and slope zero in the interval $[l_j - p_j, \theta_4]$. Scaling by d_j shows the statement. $\square$

Lemma 6. *Let $\theta' = \max\{\theta_3, \theta_4, \bar{t}_1\}$. For any job $j \in J$ and $\bar{t}_1 \in [e_j, l_j]$ the piecewise linear function $f_j(t_2) = d_j \cdot (\mu_j^{right}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta', \infty)$ decomposes into the linear function segments*

$$f_j^1(t_2) = d_j \cdot (t_2 - \theta'), \quad t_2 \in [\theta', l_j]$$
$$f_j^2(t_2) = d_j \cdot (l_j - \theta'), \quad t_2 \in [l_j, \infty).$$

Proof. Since $\bar{t}_1 \in t_1 \in [e_j, l_j]$, the function $\mu_j^{right}(\bar{t}_1, t_2)$ has slope one in the interval $[\theta', l_j]$ and slope zero in the interval $[l_j, \infty)$. The function $\mu_j(\bar{t}_1, t_2)$ is constant for all $t_2 \in [\theta', \infty)$. Hence, the function $\mu_j^{right}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2)$ has slope one in the interval $[\theta', l_j]$ and slope zero in the interval $[l_j, \infty)$. Scaling by d_j shows the statement. $\square$

Lemma 7. *Let $\theta' = \max\{\theta_3, \theta_4, \bar{t}_1\}$. For any job $j \in J$ and $\bar{t}_1 \in [e_j, l_j]$ the piecewise linear function $f_j(t_2) = -d_j \cdot (\bar{t}_1 + \mu_j(\bar{t}_1, t_2))$ is constant on $[\theta', \infty)$.*

Proof. By construction, it holds $\mu_j(\bar{t}_1, t_2) = \mu_j(\bar{t}_1, \theta')$ for all $t_2 \in [\theta', \infty)$ which is constant. Consequently, $f_j(t_2)$ is constant for all $t_2 \in [\theta', \infty)$. $\square$

A.5   Proof of Lemma 8

Lemma 8. *If the slopes of the functions (23) and (24) coincide on an interval $[\underline{t}_2, \overline{t}_2] \subset \mathbb{R}$ then both functions on the interval $[\underline{t}_2, \overline{t}_2]$ attain their maximum at the same point $(\overline{t}_1, t_2) \in \mathcal{T}$ with $t_2 \in [\underline{t}_2, \overline{t}_2]$, if the maximum exists.*

Proof. Since the slope of (23) equals the slope of (24) the quotient of the functions $\omega(\overline{t}_1, t_2) + d_j \cdot (\mu_j^{left}(\overline{t}_1, t_2) - \mu_j(\overline{t}_1, t_2))$ and $\omega(\overline{t}_1, t_2) + d_j \cdot (t_2 - \mu_j(\overline{t}_1, t_2))$ is constant for all $t_2 \in [\underline{t}_2, \overline{t}_2]$. Hence, if there exists an interval $(\overline{t}_1, t_2) \in \mathcal{T}$ with $t_2 \in [\underline{t}_2, \overline{t}_2]$ that maximizes (23) it also maximizes (24) and conversely.   □

A.6   Proof of Lemmas 9–12

Lemma 9. *For fixed value $\overline{t}_1 \in [e_j, l_j]$ the function $\mu_j^{right}(\overline{t}_1, t_2) - 2 \cdot \mu_j(\overline{t}_1, t_2)$ has slope zero for all $t_2 \in [l_j, \infty)$.*

Proof. Both functions $\mu_j^{right}(\overline{t}_1, t_2)$ and $\mu_j(\overline{t}_1, t_2)$ have slope zero in the interval $t_2 \in [l_j, \infty)$, so the stated function has slope zero in this interval.   □

Lemma 10. *For fixed value $\overline{t}_1 \in [e_j, \theta_2]$ the function $t_2 - \mu_j^{left}(\overline{t}_1, t_2)$ has slope one the interval $t_2 \in [e_j + p_j, \theta_4]$.*

Proof. The function $\mu_j^{left}(\overline{t}_1, t_2)$ has slope zero in the interval $t_2 \in [e_j + p_j, \theta_4]$, so $t_2 - \mu_j^{left}(\overline{t}_1, t_2)$ has slope one in the interval $t_2 \in [e_j + p_j, \theta_4]$.   □

Lemma 11. *For fixed value $\overline{t}_1 \in [e_j, \theta_2]$ the function $\mu_j^{right}(\overline{t}_1, t_2) - 2 \cdot \mu_j(\overline{t}_1, t_2)$ has slope one in the interval $[\max\{\theta_3, \theta_4\}, l_j]$.*

Proof. The function $\mu_j^{right}(\overline{t}_1, t_2)$ has slope one and the function $\mu_j(\overline{t}_1, t_2)$ has slope zero in the interval $[\max\{\theta_3, \theta_4\}, l_j]$, so the stated function has slope one.   □

Lemma 12. *For fixed value $\overline{t}_1 \in [e_j, \theta_2]$ the function $t_2 - \mu_j^{left}(\overline{t}_1, t_2)$ has slope zero in the interval $[t_1, \theta_2]$.*

Proof. The function $\mu_j^{left}(\overline{t}_1, t_2)$ has slope one in the interval $[t_1, \theta_2]$, therefore $t_2 - \mu_j^{left}(\overline{t}_1, t_2)$ has slope zero.   □

B   Algorithms

Notes for the algorithms:

- $(j, t_2, \tau_2) \in O_3(t_1) \iff (j, t_2 + t_1, \tau_2) \in O_3$
- the computation of the energy overloads $\omega(t_1, t_2)$ is analogous to the checker of Baptiste et al. [5] and involves dynamic slope updates

---

Algorithm 1. Dynamic $\mathcal{O}(n^2)$ interval checker

Input: CuSP instance
Output: false, if there exists a time interval positive overload
1 $O_1 \leftarrow \{(i, e_i, T_1) \mid i \in J\} \cup \{(i, l_i - p_i, T_1^M) \mid i \in J^M\}$
2 $O_2 \leftarrow \{(i, l_i, T_2) \mid i \in J\}$
3 $O_3 \leftarrow \emptyset$
4 $t_1^{old} \leftarrow \infty$

5 **for** $(i, t_1, \tau_1) \in O_1$ *non-decreasing* t_1 **do**
6   **if** $t_1 = t_1^{old}$ **then** *continue*
7      $t_1^{old} \leftarrow t_1$
8   **for** $curr = (j, t_2, \tau_2) \in O_2 \cup O_3(t_1)$ *non-decreasing* $t_2 > t_1$ **do**
9   **if** $\omega(t_1, t_2) > 0$ **then return** $false$ // see Baptiste et al.
10   **if** $t_1 \geq e_j + p_j$ **then**   // update list elements of O_2 and O_3
11   **if** $curr \in O_2$ **then** $O_2.delete(curr)$
12   **else** $O_3.delete(curr)$
13   **else if** $\tau_2 = T_2 \wedge t_1 = e_j$ **then**
14           $O_2.delete(curr)$
15   **if** $e_j + p_j = l_i$ **then** $O_2.insert(j, t_2, T_2^M)$
16   **else** $O_3.insert(i, e_i + l_i, T_3)$
17   **else if** $\tau_2 = T_3 \wedge t_1 = l_j - p_j$ **then**
18           $O_3.delete(curr)$
19           $O_2.insert(j, t_2, T_2^M)$

20 **return** $true$

Algorithm 2. Reduced $\mathcal{O}(n^3)$ propagator

Input: CuSP instance
Output: false, if there exists a time interval with positive overload,
            otherwise propagate earliest start and latest completion times
$O_1 \leftarrow \{(j, e_j, T_1) \mid j \in J\} \cup \{(j, l_j - p_j, T_1^M) \mid j \in J\}$
$O_2 \leftarrow \{(j, l_j, T_2) \mid j \in J\} \cup \{(j, e_j + p_j, T_2^M) \mid j \in J\}$
$O_3 \leftarrow \{(j, e_j + l_j, T_3) \mid j \in J\}$
$t_1^{old} \leftarrow \infty$

1 **for** $(i, t_1, \tau_1) \in O_1$ *non-decreasing* t_1 **do**
2   **if** $t_1 = t_1^{old}$ **then** *continue*
3      $t_1^{old} = t_1$

4   **for** $curr = (j, t_2, \tau_2) \in O_2 \cup O_3(t_1)$ *non-decreasing* $t_2 > t_1$ **do**
5   **if** $\omega(t_1, t_2) > 0$ **then return** $false$ // see Baptiste et al.

        // left- and right-shift propagations
6   **for** $j \in J$ **do**
7   **if** $\omega(t_1, t_2) + d_j \cdot (\mu_j^{left}(t_1, t_2) - \mu_j(t_1, t_2)) > 0$ **then**
8              $e_j \leftarrow t_2 - \mu_j(t_1, t_2) + \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil$
9   **if** $\omega(t_1, t_2) + d_j \cdot (\mu_j^{right}(t_1, t_2) - \mu_j(t_1, t_2)) > 0$ **then**
10             $l_j \leftarrow t_1 + \mu_j(t_1, t_2) - \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil$

11 **return** $true$

Algorithm 3. $\mathcal{O}(n^2 \log n)$ Sweep Line Propagator

Input: CuSP instance
Output: false, if there exists a time interval with positive overload,
        otherwise propagate earliest start and latest completion times

1 $O_1 \leftarrow \{(j, e_j, T_1) \mid i \in J\} \cup \{(j, l_j - p_j, T_1^M) \mid j \in J\}$
2 $O_2 \leftarrow \{(j, l_j, T_2) \mid i \in J\} \cup \{(j, e_j + p_j, T_2^M) \mid j \in J\}$
3 $O_3 \leftarrow \{(j, e_j + l_j, T_3) \mid i \in J\}$
4 $t_1^{old} \leftarrow \infty$
5 $e'_j \leftarrow e_j \quad \forall j \in J$
6 $l'_j \leftarrow l_j \quad \forall j \in J$

7 **for** $(i, t_1, \tau_1) \in O_1$ *non-decreasing* t_1 **do**
8   **if** $t_1 = t_1^{old}$ **then** *continue*
9 $t_1^{old} = t_1$
10 $t_2^{old} \leftarrow \infty$

11   **for** $curr = (j, t_2, \tau_2) \in O_2 \cup O_3(t_1)$ *non-decreasing* $t_2 > t_1$ **do**
12   **if** $\omega(t_1, t_2) > 0$ **then** **return** *false*   `// see Baptiste et al.`
13   **if** $t_2 \neq t_2^{old}$ **then**
14 $\mathcal{P} \leftarrow \mathcal{P} \cup \{(t_2, \omega(t_1, t_2))\}$ `// add points`
15 $t_2^{old} \leftarrow t_2$

16 $\mathcal{L} \leftarrow initializeLineSegments(t_1)$ `// add line segments`
17   **if** $\mathcal{P} \neq \emptyset \wedge \mathcal{L} \neq \emptyset$ **then**
18 $sweepLinePropagation(\mathcal{P}, \mathcal{L}, e', l', t_1)$ `// propagate`

19 **for** $j \in J$ **do**
20   **if** $e'_j > e_j$ **then** $e_j = e'_j$
21   **if** $l'_j < l_j$ **then** $l_j = l'_j$
22   **if** $e_j + p_j > l_j$ **then** **return** *false*

23 **return** *true*

Algorithm 4. Initialize Line Segments

Input: jobs J, lower interval limit t_1
Output: set of line segments $\mathcal{L}$
$\mathcal{L} \leftarrow \emptyset$

1 for $j \in J$ **do**
2 $\theta_1 = \max\{e_j, t_1\}$
3 $\theta_2 = \min\{e_j + p_j, l_j - p_j\}$
4 $\theta_3 = \max\{e_j + p_j, l_j - p_j\}$
5 $\theta_4 = \min\{l_j, e_j + l_j - t_1\}$

    `// add lines`
    `// format:  (job,` $\underline{x}_l, \overline{x}_l$ `, slope, left-/right-shift)`
6   **if** $t_1 \in [e_j, \theta_2)$ **then**
       `// exact`
7 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, l_j, \infty, 0, right)$ `// region A`
8 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_1, \theta_2, d_j, left)$ `// region C`
       `// this line segment is always added`
9 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_2, \theta_3, 0, left)$ `// region B or C`

       `// non-exact`
       `// add slopes of overload and update function`
10   **if** $e_j + p_j < l_j - p_j$ **then**
11 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_2, \theta_3, d_j, left)$ `// region B`
12 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_3, \theta_4, 0, left)$ `// region B`
13 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_3, \theta_4, -d_j, left)$ `// region B`
14 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_4, l_j, d_j, right)$ `// region B'`
15 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_4, l_j, 0, right)$ `// region B'`

16   **else if** $t_1 \in [\theta_2, \theta_3)$ **then**
       `// exact`
17 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, l_j, \infty, 0, right)$ `// region A`

       `// non-exact`
       `// add slopes of overload and update function`
18   **if** $e_j + p_j < l_j - p_j$ **then**
19 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_3, l_j, d_j, right)$ `// region B'`
20 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_3, l_j, 0, right)$ `// region B'`

21   **else if** $t_1 \in [\theta_3, l_j)$ **then**
       `// exact`
22 $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, l_j, \infty, 0, right)$ `// region A`

23 return $\mathcal{L}$

Algorithm 5. Sweep Line Propagation

Input: points $\mathcal{P}$, line segments $\mathcal{L}$, earliest start times e'_j, latest completion times l'_j, lower interval limit t_1

Output: updated start times e'_j and completion times l'_j for all $j \in J$

1   $a_0 \leftarrow \min\{a_l \mid l \in \mathcal{L}\}$ `// start slope`

2   $H \leftarrow \emptyset$ `// empty event heap`

3   **for** $l \in \mathcal{L}$ **do**   `// fill heap with evaluation events`

4      $H.insert(a_l, l, evaluation)$

5   $B \leftarrow initializeIntervalTree(\mathcal{P}, a_0, H)$ `// build interval tree`

6   **while** $\neg H.empty$ **do**   `// sweep over all events`

7      $event \leftarrow H.extractMin$

8   **if** $event.type = evaluation$ **then**

9          $l \leftarrow event.getLine$

10         $j \leftarrow l.job$

11         $(t_2, \omega(t_1, t_2)) \leftarrow B.evaluate(l)$ `// evaluate`

12   **if** $l.shift = left \wedge \omega(t_1, t_2) + d_j \cdot (\mu_j^{left}(t_1, t_2) - \mu_j(t_1, t_2)) > 0$ **then**

13             $update \leftarrow t_2 - \mu_j(t_1, t_2) + \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil$

14   **if** $update > e'_j$ **then** $e'_j \leftarrow update$

15   **else if**

            $l.shift = right \wedge \omega(t_1, t_2) + d_j \cdot (\mu_j^{right}(t_1, t_2) - \mu_j(t_1, t_2)) > 0$ **then**

16             $update \leftarrow t_1 + \mu_j(t_1, t_2) - \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil$

17   **if** $update < l'_j$ **then** $l'_j \leftarrow update$

18   **else if** $event.type = resolve$ **then**

19          $B.resolve(event.getTreeNode, H)$ `// resolve`

20   **return** e', l'

Algorithm 6. Initialize Interval Tree

Input: set of points $\mathcal{P}$, initial slope a_0 event heap H
Output: interval tree B

1   $B \leftarrow$ binary interval tree with $2^{\lceil \log_2 \mathcal{P} \rceil + 1}$ empty nodes
2   $\mathcal{P} \leftarrow \mathcal{P}$ sorted by x_q first then by y_q for all $q \in \mathcal{P}$ in non-decreasing order
3   $q \leftarrow \mathcal{P}.first$

    // leaf nodes
4   **for** $v \in V_B^{leaf}$ *from left to right* $\wedge\, q \neq \mathcal{P}.end$ **do**
5      $[\underline{x}_v, \overline{x}_v] = [x_q, x_q]$
6      $\pi_v \leftarrow v$
7      $\alpha_v \leftarrow \infty$
8      $\beta_v \leftarrow \infty$
9      $q \leftarrow \mathcal{P}.next$

    // non-leaf nodes
10   **for** $v \in V_B^{noleaf}$ *sorted from bottom to top and from left to right* **do**
11   **if** $v.left = null$ **then** *continue*
12   **if** $v.right \neq null$ **then**
13        $[\underline{x}_v, \overline{x}_v] \leftarrow [\underline{x}_{v.left}, \overline{x}_{v.right}]$
14        $\beta_v \leftarrow \min\{\alpha_{v.left}, \beta_{v.left}, \alpha_{v.right}, \beta_{v.right}\}$
15   **if** $a_0 \cdot \pi_{v.left}.x + \pi_{v.left}.y > a_0 \cdot \pi_{v.right}.x + \pi_{v.right}.y$ **then**
16          $\pi_v \leftarrow \pi_{v.left}$
17   **else** $\pi_v \leftarrow \pi_{v.right}$
18   **if** $\pi_{v.left}.x \neq \pi_{v.right}.x$ **then**
19          $val \leftarrow \frac{\pi_{v.right}.y - \pi_{v.left}.y}{\pi_{v.left}.x - \pi_{v.right}.x}$
20   **if** $val > a_0$ **then**
21            $\alpha_v \leftarrow val$
22   **if** $\alpha_v < \beta_v$ **then**
23              $H.insert(\alpha_v, v, resolve)$ // add resolve event
24   **else** $\alpha_v \leftarrow \infty$
25   **else** $\alpha_v \leftarrow \infty$
26   **else**
27        $[\underline{x}_v, \overline{x}_v] \leftarrow [\underline{x}_{v.left}, \overline{x}_{v.left}]$
28        $\pi_v \leftarrow \pi_{v.left}$
29        $\alpha_v \leftarrow \infty$
30        $\beta_v \leftarrow \min\{\alpha_{v.left}, \beta_{v.left}\}$

31 **return** B

Algorithm 7. Evaluate

Input: interval tree B, line segment $l \in \mathcal{L}$
Output: point $q^* = (x_{q^*}, y_{q^*}) \in \mathcal{P}$ with

$$q^* = \arg\max_{q \in \mathcal{P}: x_q \in [\underline{x}_l, \overline{x}_l]} a_l \cdot x_q + y_q$$

1 $max \leftarrow -\infty$
2 $S \leftarrow \emptyset$ // empty stack
3 $S.push(B.root)$

4 **while** $\neq S.empty$ **do**
5 $\quad$ $v \leftarrow S.pop$
6 $\quad$ **if** $v = null$ **then** $\;continue$

7 $\quad$ **if** $[\underline{x}_v, \overline{x}_v] \subseteq [\underline{x}_l, \overline{x}_l]$ **then**
8 $\quad\quad$ $val \leftarrow a_l \cdot \pi_v.x + \pi_v.y$
9 $\quad\quad$ **if** $val > max$ **then**
10 $\quad\quad\quad$ $max \leftarrow val$
11 $\quad\quad\quad$ $q^* \leftarrow \pi_v$

12 $\quad$ **else if** $[\underline{x}_v, \overline{x}_v] \cap [\underline{x}_l, \overline{x}_l] \neq \emptyset$ **then**
13 $\quad\quad$ $S.push(v.left)$
14 $\quad\quad$ $S.push(v.right)$

15 **return** q^*

Algorithm 8. Resolve

Input: interval tree B, interval tree node $v \in V_B$, event heap H
Output: updates the interval tree and propagates changes to the root
$\quad\quad\quad\quad$ node, add new resolve events to heap
// resolve current node
1 $\pi_v \leftarrow \pi_{v.right}$
2 $\alpha_v \leftarrow \infty$
3 $\beta_v \leftarrow \min\{\alpha_{v.right}, \beta_{v.right}\}$

// propagate changes to root node
4 $prev \leftarrow v$
5 $v \leftarrow v.parent$

6 **while** $prev \neq T.root$ **do**
7 $\quad$ **if** $v.right \neq null$ **then**
8 $\quad\quad$ **if** $\alpha_v < \infty$ **then**
9 $\quad\quad\quad$ **if** $\pi_{v.right}.x \neq \pi_{v.left}.x$ **then**
10 $\quad\quad\quad\quad$ $\alpha_v \leftarrow \frac{\pi_{v.left}.y - \pi_{v.right}.y}{\pi_{v.right}.x - \pi_{v.left}.x}$
11 $\quad\quad\quad$ **else** $\alpha_v \leftarrow \infty$
12 $\quad\quad\quad$ $\beta_v \leftarrow \min\{\alpha_{v.left}, \beta_{v.left}, \alpha_{v.right}, \beta_{v.right}\}$

13 $\quad\quad$ **else** $\beta_v \leftarrow \min\{\alpha_{v.right}, \beta_{v.right}\}$
14 $\quad\quad$ **if** $\alpha_v < \beta_v$ **then**
15 $\quad\quad\quad$ $H.insert(\alpha_v, v, resolve)$ // add resolve event

16 $\quad$ **else** $\beta_v \leftarrow \min\{\alpha_{v.left}, \beta_{v.left}\}$
17 $\quad$ $prev \leftarrow v$
18 $\quad$ $v \leftarrow v.parent$

References

1. Artigues, C., Lopez, P.: Energetic reasoning for energy-constrained scheduling with a continuous resource. J. Sched. **18**(3), 225–241 (2015)
2. Bonifas, N.: A $O(n^2 log(n))$ propagation for the energy reasoning. In: Conference Paper, Roadef 2016 (2016)
3. Derrien, A., Petit, T.: A new characterization of relevant intervals for energetic reasoning. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 289–297. Springer, Heidelberg (2014)
4. Berthold, T., Heinz, S., Schulz, J.: An approximative criterion for the potential of energetic reasoning. In: Marchetti-Spaccamela, A., Segal, M. (eds.) TAPAS 2011. LNCS, vol. 6595, pp. 229–239. Springer, Heidelberg (2011)
5. Baptiste, P., Le Pape, C., Nuijten, W.: Satisfiability tests and time bound adjustments for cumulative scheduling problems. Ann. Oper. Res. **92**, 305–333 (1999)
6. Baptiste, P., Le Pape, C., Nuijten, W.: Applying Constraint Programming to Scheduling Problems, vol. 39. Springer Science & Business Media (2012)
7. Vilím, P.: Edge finding filtering algorithm for discrete cumulative resources in $\mathcal{O}(kn \log n)$. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 802–816. Springer, Heidelberg (2009)
8. Vilím, P.: Timetable edge finding filtering algorithm for discrete cumulative resources. In: Achterberg, T., Beck, J.C. (eds.) CPAIOR 2011. LNCS, vol. 6697, pp. 230–245. Springer, Heidelberg (2011)
9. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints **16**(3), 250–282 (2011)
10. Schutt, A., Wolf, A.: A new $\mathcal{O}(n^2 \log n)$ not-first/not-last pruning algorithm for cumulative resource constraints. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 445–459. Springer, Heidelberg (2010)
11. Schutt, A., Feydy, T., Stuckey, P.J.: Explaining time-table-edge-finding propagation for the cumulative resource constraint. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013. LNCS, vol. 7874, pp. 234–250. Springer, Heidelberg (2013)
12. Ouellet, P., Quimper, C.-G.: Time-table extended-edge-finding for the cumulative constraint. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 562–577. Springer, Heidelberg (2013)
13. Kameugne, R., Fotso, L.P., Scott, J., Ngo-Kateu, Y.: A quadratic edge-finding filtering algorithm for cumulative resource constraints. Constraints **19**(3), 243–269 (2014)
14. Mercier, L., Van Hentenryck, P.: Edge finding for cumulative scheduling. INFORMS J. Comput. **20**(1), 143–153 (2008)
15. Letort, A., Beldiceanu, N., Carlsson, M.: A scalable sweep algorithm for the *cumulative* constraint. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 439–454. Springer, Heidelberg (2012)
16. Kolisch, R., Sprecher, A.: PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program. Eur. J. Oper. Res. **96**(1), 205–216 (1997)
17. Godard, D., Laborie, P., Nuijten, W.: Randomized large neighborhood search for cumulative scheduling. In: ICAPS, vol. 5, pp. 81–89, June 2005
18. HP Peter Stuckey. http://people.eng.unimelb.edu.au/pstuckey/rcpsp/

Efficient Filtering for the Unary Resource with Family-Based Transition Times

Sascha Van Cauwelaert[1], Cyrille Dejemeppe[1(✉)], Jean-Noël Monette[2], and Pierre Schaus[1]

[1] UCLouvain, ICTEAM, Louvain-la-Neuve, Belgium
{sascha.vancauwelaert,cyrille.dejemeppe,pierre.schaus}@uclouvain.be
[2] Tacton Systems, Stockholm, Sweden
jean-noel.monette@tacton.com

Abstract. We recently proposed an extension to Vilím's propagators for the unary resource constraint in order to deal with sequence-dependent transition times. While it has been shown to be scalable, it suffers from an important limitation: when the transition matrix is sparse, the additional filtering, as compared to the original from Vilím's algorithm, drops quickly. Sparse transition time matrices occur especially when activities are grouped into families with zero transition times within a family. The present work overcomes this weakness by relying on the transition times between families of activities. The approach is experimentally evaluated on instances of the Job-Shop Problem with Sequence Dependent Transition Times. Our experimental results demonstrate that the approach outperforms existing ones in most cases. Furthermore, the proposed technique scales well to large problem instances with many families and activities.

Keywords: Constraint programming · Scheduling · Job-Shop · Sequence-dependent transition times · Family · Global constraint · Traveling Salesman Problem · Dynamic programming · Lower bound

1 Introduction

Unary resources with sequence-dependent transition times (also called set-up times) for non-preemptive activities are very frequent in real-life scheduling problems. A first example is the quay crane scheduling in container terminals [20], where the crane is modeled as a unary resource and transition times represent the moves of the crane on the rail between positions where it needs to load or unload containers. A second example is the continuous casting scheduling problem [9], where a set-up time is required between production programs.

Although efficient propagators have been designed for the standard (UR) [16], transition time constraints between activities generally make the problem harder to solve because the existing propagators do not take them into account. We recently introduced in [5] a propagator for the unary resource constraint with

---

transition times (URTT) as an extension to Vilím's algorithms, in order to strengthen the filtering in the presence of transition times.

Unfortunately, the additional filtering quickly drops in the case of a sparse transition time matrix, which typically occurs when activities are grouped into families with zero transition times within a family. The reason for a weak filtering with sparse matrices is that it is based on a shortest path problem with free starting and ending nodes and a fixed number of edges. The length of this shortest path drops in the case of zero transition times.

The main contribution of the present article is to introduce adapted filtering rules considering the families. The new propagator also relies on a shortest path problem but over a different underlying graph. The main asset of our approach is its scalability: we obtain an important amount of filtering while keeping a low time complexity of $O(n \log(n) \log(f))$, for n activities and f families. In general $f \ll n$, hence the theoretical complexity is very close to the one of the propagators in [5,16]. The filtering is experimentally tested on instances of the Job-Shop Problem with Sequence Dependent Transition Times (JSPSDTT), although it can be used for any type of problems, e.g., with other kinds of objective function than the makespan minimization. The results show that our propagator improves the resolution time over existing approaches and is more scalable.

The paper starts by providing the background for the considered problems in Sect. 2. The work on the URTT propagator [5] is also briefly recalled and its limitations are highlighted. Then, Sect. 3 presents the stronger filtering making use of the families. Section 4 reviews alternative approaches and Sect. 5 compares the results of the different approaches.

2   Background

Non-preemptive scheduling problems are usually modeled in constraint programming (CP) by associating three variables to each activity A_i: s_i, c_i, and p_i representing respectively the starting time, completion time, and processing time of A_i. These variables are linked together by the following relation: $s_i + p_i = c_i$. Depending on the problem, the scheduling of the activities can be restricted by the availability of different kinds of resources required by the activities. In this work, we are interested in the unary resource (sometimes referred to as a machine or a disjunctive resource) and the propagators associated to one unary resource. Let T be the set of activities requiring the considered unary resource. The unary resource constraint prevents any two activities in T to overlap in time:

$$\forall A_i, A_j \in T : A_i \neq A_j \implies (c_i \leq s_j) \vee (c_j \leq s_i)$$

The unary resource can be generalized by requiring transition times between activities. The transition times are described by a square matrix $\mathcal{TT}$ in which $tt_{i,j}$, the entry at line i and column j, represents the minimum amount of time that must occur between the activities A_i and A_j when A_i directly precedes A_j. We assume

that transition times respect the triangular inequality. That is, inserting an activity between two activities never decreases the transition time between these two activities: $\forall A_i, A_j, A_k \in T : tt_{i,j} \leq tt_{i,k} + tt_{k,j}$.

The unary resource with transition times constraint imposes the following relation:

$$\forall A_i, A_j \in T : A_i \neq A_j \implies (c_i + tt_{i,j} \leq s_j) \vee (c_j + tt_{j,i} \leq s_i) \qquad (1)$$

The earliest starting time of an activity A_i is denoted est_i and its latest starting time is denoted lst_i. The domain of s_i is thus the interval $[est_i; lst_i]$. Similarly the earliest completion time of A_i is denoted ect_i and its latest completion time is denoted lct_i. The domain of c_i is thus the interval $[ect_i; lct_i]$. These definitions can be extended to a set of activity Ω. For instance, est_Ω is the earliest time when any activity in Ω can start and ect_Ω is the earliest time when all activities in Ω can be completed. We also define $p_\Omega = \sum_{A_j \in \Omega} p_j$ to be the sum of the processing times of the activities in Ω. While one can directly compute $est_\Omega = \min \{est_j | A_j \in \Omega\}$ and $lct_\Omega = \max \{lct_j | A_j \in \Omega\}$, it is NP-hard to compute the exact values of ect_Ω and lst_Ω [16]. Instead, one usually computes a lower bound for ect_Ω and an upper bound for lst_Ω. The propagators of [5,16] allow to compute efficiently such lower bounds, but have limitations in the presence of family-based transition times.

2.1 Propagator for the Unary Resource

The filtering rules presented in [16] for the UR constraint fall in several categories known as Overload Checking (OC), Detectable Precedences (DP), Not-First/Not-Last (NF/NL), and Edge Finding (EF). The implementation of these filtering rules runs in $\mathcal{O}(n \log(n))$, with $n = |T|$. It relies on an efficient computation of a lower bound ect_Ω^{LB0} of the earliest completion time of a set of activities $\Omega \subseteq T$, defined as:

$$ect_\Omega^{LB0} = \max_{\Omega' \subseteq \Omega} \{est_{\Omega'} + p_{\Omega'}\} \qquad (2)$$

The rules OC, DP, and NF/NL, rely on the so-called Θ-tree data structure, while EF relies on the Θ-Λ-tree data structure. The Θ-tree and the Θ-Λ-tree are used to compute efficiently and incrementally ect_Θ^{LB0} on a set of activities Θ. For instance, the OC rule is used to detect when $ect_\Theta^{LB0} > lct_\Theta$ for any $\Theta \subseteq T$, which triggers a failure. We refer the reader to [16] for a detailed description of this and the other rules. The following example illustrates the missed filtering for UR when it does not consider the transition times globally.

Example 1. *Consider a set of 3 activities $\Omega = \{A_1, A_2, A_3\}$ as shown in Fig. 1. Consider also, for simplicity, that all pairs of activities from Ω have the same transition time $tt_{i,j} = 3$. The OC rule detects a failure when $ect_\Omega^{LB0} > lct_\Omega$. The filtering as described in [16] computes:*

$$ect_\Omega^{LB0} = est_\Omega + \sum_{A_i \in \Omega} p_i = 0 + 5 + 5 + 3 = 13$$

As we have $lct_\Omega = \max_{A_i \in \Omega} lct_i = lct_2 = 17$, the OC rule from [16], combined with the transition times binary decomposition (Eq. (1)), does not detect a failure. However, as there are 3 activities in Ω, at least two transitions occur between these activities and it is actually not possible to find a feasible schedule. Indeed, taking these transition times into account, one could compute $ect_\Omega = 13 + 2 \cdot tt_{i,j} = 13 + 2 \cdot 3 = 19 > 17 = lct_\Omega$, and thus detect the failure.
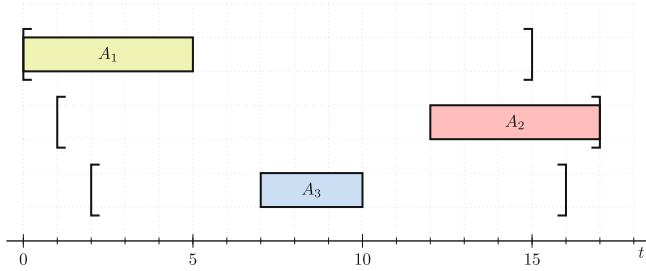
Fig. 1. Example illustrating the missed failure detection of OC when not considering transition times.

2.2 Propagator for the Unary Resource with Transition Times

In [5], we extended Vilím's work [16] to the case of the unary resource with transition times constraint. By extending the Θ-tree and the Θ-Λ-tree to take the transition times into account to compute a lower bound of ect_Θ, we could strengthen the filtering without increasing the time and space complexities.[1]

Let Π_Ω be the set of all possible permutations of activities in Ω. For a given permutation $\pi \in \Pi_\Omega$, where $\pi(i)$ is the activity taking place at position i, we can define the total time spent by transition times, tt_π, as follows:

$$tt_\pi = \sum_{i=1}^{|\Omega|-1} tt_{\pi(i),\pi(i+1)}$$

A lower bound for ect_Ω can then be defined as:

$$ect_\Omega^{LB1} = \max_{\Omega' \subseteq \Omega} \left\{ est_{\Omega'} + p_{\Omega'} + \min_{\pi \in \Pi_{\Omega'}} tt_\pi \right\} \tag{3}$$

Unfortunately, computing this value is NP-hard as computing the optimal permutation $\pi \in \Pi$ minimizing tt_π amounts to solving a TSP. Since embedding an exponential algorithm in a propagator is generally impractical, a looser lower bound can be used instead.

---

[1] Strictly speaking, the propagators are not sufficient to prove Eq. (1) is respected, so the binary propagators for Eq. (1) must remain active to ensure correctness.

More precisely, for each possible subset of cardinality $k \in \{0, \ldots, n\}$, we compute the smallest transition time permutation of size k on the set T of all activities requiring the resource:

$$\underline{tt}(k) = \min_{\{\Omega' \subseteq T: \; |\Omega'| = k\}} \left\{ \min_{\pi \in \Pi_{\Omega'}} tt_\pi \right\} \qquad (4)$$

For each k, the lower bound computation thus requires one to find the shortest node-distinct $(k-1)$-edge path between any two nodes, which is also NP-hard as it can be casted into a resource-constrained shortest path problem. We proposed in [5] various lower bounds to achieve the pre-computation in polynomial time. Our final lower bound formula for the earliest completion time of a set of activities, making use of pre-computed lower-bounds of transition times, is:

$$ect_\Omega^{LB2} = \max_{\Omega' \subseteq \Omega} \left\{ est_{\Omega'} + p_{\Omega'} + \underline{tt}(|\Omega'|) \right\} \qquad (5)$$

The different lower bounds of ect_Ω can be ordered as follows:

$$ect_\Omega^{LB0} \le ect_\Omega^{LB2} \le ect_\Omega^{LB1} \le ect_\Omega$$

In order to compute ect_Ω^{LB2} incrementally, adapted versions of the Θ-tree and the Θ-Λ-tree were introduced in [5].

Limitation. An important limitation of this approach arises in the context of *sparse* transition matrices. Indeed, when there exists a node-distinct path with K zero-transition edges, we have: $\underline{tt}(k) = 0 \; \forall k \in \{1, \ldots, K\}$. The pruning achieved by the propagator is then equivalent to the one of the original algorithms from Vilím [16], which has been shown to perform poorly when transition times are involved (see [5]). To cope with that problem, we propose to reason with *families* of activities, as described in the next section.

Example 2. *Consider again the three activities $\Omega = \{A_1, A_2, A_3\}$ shown in Fig. 1 with A_1 belonging to family F_1, A_2 to family F_2, and A_3 to family F_3. The transition times are equal to 3 between activities from different families and equal to 0 between activities of the same family. Assume that 3 additional activities (not represented) also belong to family F_1. Because the transition times between any pair of activity from a same family is 0, we have that $\underline{tt}(2) = \underline{tt}(3) = 0$ and $ect_\Omega^{LB2} = 13 = ect_\Omega^{LB0}$, hence the OC of [5] is unable to detect the failure.*

3   Filtering with Families of Activities

When transition times are present, it is often the case that activities are grouped in families on which the transition times are expressed. Formally, we denote by $F(A_i)$ the family of activity A_i and by $\mathcal{F}$ the set of all families. In a family-based setting, the transition times are described as a square matrix $\mathcal{TT}^{\mathcal{F}}$ of size $|\mathcal{F}|$. The transition time between two activities A_i and A_j is the transition time

between their respective families $F(A_i)$ and $F(A_j)$, and it is zero if $F(A_i) = F(A_j)$:

$$\forall A_i, A_j \in T : tt_{i,j} = tt^{\mathcal{F}}_{F(A_i),F(A_j)} \wedge \left(F(A_i) = F(A_j) \implies tt^{\mathcal{F}}_{F(A_i),F(A_j)} = 0 \right) \tag{6}$$

The matrix $\mathcal{TT}^{\mathcal{F}}$ is smaller and less sparse than the original matrix $\mathcal{TT}$.

To cope with the limitations highlighted in Sect. 2.2, we adapt in the present section Vilím's propagators [16] to include transition times between families while keeping a low time complexity: $\mathcal{O}(n \log(n) \log(|\mathcal{F}|))$, where $n = |T|$. To do so, we adapt the algorithms and the Θ-tree and Θ-Λ-tree data structures in a way similar to [5]: the number of different families present in a set Ω of activities is used instead of the cardinality of Ω. Counting the number of families results in non-zero lower bounds even for small sets, assuming that there are no zero transition times between families. Formally, Eq. (5) is replaced by:

$$ect^{LB3}_{\Omega} = \max_{\Omega' \subseteq \Omega} \left\{ est_{\Omega'} + p_{\Omega'} + \underline{tt}(|F_{\Omega'}|) \right\} \tag{7}$$

where $F_{\Omega} = \{F(A_i) \mid A_i \in \Omega\}$. The term $\underline{tt}(|F_{\Omega'}|)$ in Eq. (7) is pre-computed using the lower bounds introduced in [5] for $\underline{tt}(|\Omega'|)$, but using $\mathcal{TT}^{\mathcal{F}}$ instead of $\mathcal{TT}$.

Lemma 1. *In the presence of families, $ect^{LB2}_{\Omega} \leq ect^{LB3}_{\Omega}$.*

Proof (Sketch). $\mathcal{TT}^{\mathcal{F}}$ induces a graph that is isomorphic to a subgraph of the graph induced by $\mathcal{TT}$ and any (shortest) path induced by $\mathcal{TT}^{\mathcal{F}}$ has a corresponding valid path induced by $\mathcal{TT}$. □

Computing ect^{LB3}_{Ω} requires some careful adaptations to the algorithms (Sect. 3.1) and data structures (Sects. 3.2 and 3.3).

3.1 Adapting the Algorithms

We adapt the original algorithms of [16] in order to consider transition times. While most of the modifications impact the underlying Θ-tree and Θ-Λ-tree data structures, the filtering rules are also slightly adapted. This is done in a similar manner to [5], but reasoning with F_{Ω} instead of Ω.

For instance, in the original algorithms of [16] (OC, DP, NF/NL and EF), if the activity i is detected as having to take place after all activities in a set Θ, the following update rule can be applied: $est_i \leftarrow \max \{est_i, ect^{LB0}_{\Theta}\}$. As transition times are involved, we can replace ect^{LB0}_{Θ} by ect^{LB3}_{Θ} but, additionally, the minimal transition from any family $F_j \in F_{\Theta}$ to the family $F(A_i)$ should also be added as it was not taken into account in the computation of ect^{LB3}_{Θ}. This transition is the minimal one from any family to $F(A_i)$, because we do not know which activity will be just before A_i in the final schedule. The update rule becomes:

$$est_i \leftarrow \max \left\{ est_i, ect^{LB3}_{\Theta} + \min_{F_j \in F_{\Theta}} tt^{\mathcal{F}}_{F_j, F(A_i)} \right\}$$

An analogous reasoning can be applied to the rule updating the *lct* of an activity. Finally, notice that as in [5], the transition times binary decomposition from Eq. (1) must be added to the model in order to ensure correctness. Indeed, ect_{Θ}^{LB3} contains only a lower bound of the total transition time in Θ. The propagators based on ect_{Θ}^{LB3} are thus not sufficient to ensure correctness.

3.2   Extending the Θ-tree with Families

A Θ-tree is a balanced binary tree in which each leaf represents an activity from a set Θ and internal nodes gather information about the set of activities represented by the leaves under this node, denoted $Leaves(v)$. We write $l(v)$ for the left child of v and $r(v)$ for the right one. Leaves are ordered in non-decreasing order of the *est* of the activities: for two activities A_i and A_j, if $est_i < est_j$, then the leaf representing A_i is at the left of the leaf representing A_j.

The main value stored in a node v is the lower bound of $ect_{Leaves(v)}$, denoted ect_v. To be able to compute this value incrementally upon insertion or deletion of an activity in the Θ-tree, one needs to maintain additional values.

In [16], Vilím has shown that, by defining $ect_v = ect_{Leaves(v)}^{LB0}$, it suffices to store additionally $p_v = p_{Leaves(v)}$. In a leaf v representing an activity A_i, one can compute $p_v = p_i$ and $ect_v = ect_i$. In an internal node v, one can compute:

$$p_v = p_{l(v)} + p_{r(v)}$$
$$ect_v = \max\left\{ect_{r(v)}, ect_{l(v)} + p_{r(v)}\right\}$$

Hence, the values only depend on the values stored in the two children.

In this work, we would like instead to define $ect_v = ect_{Leaves(v)}^{LB3}$ in order to take family-based transition times into account. As this value cannot easily be computed incrementally, we compute a lower bound, denoted ect_v^*. In addition to ect_v^*, one needs to store not only p_v, but also $F_v = F_{Leaves(v)}$, the set of the families of the activities in $Leaves(v)$. In a leaf v representing an activity A_i, one can compute $p_v = p_i$, $ect_v^* = ect_i$, and $F_v = \{F(A_i)\}$. In an internal node v, one can compute:

$$p_v = p_{l(v)} + p_{r(v)}$$
$$F_v = F_{l(v)} \cup F_{r(v)}$$
$$ect_v^* = \max \begin{cases} ect_{r(v)}^* \\ ect_{l(v)}^* + p_{r(v)} + \underline{tt}\big(\big|F_{r(v)} \setminus F_{l(v)}\big| + eI\left(F_{l(v)}, F_{r(v)}\right)\big) \end{cases}$$

where $eI\left(F_A, F_B\right)$ is equal to 1 if $(F_A \cap F_B) = \emptyset$, and to 0 otherwise.

Lemma 2. $ect_v^* \leq ect_{Leaves(v)}^{LB3}$

Proof (Sketch). By induction. If v is a leaf representing activity A_i, then $ect_v^* = ect_i = ect_{\{A_i\}}^{LB3}$. Otherwise, our induction hypothesis is that $ect_{l(v)}^* \leq ect_{Leaves(l(v))}^{LB3}$ and $ect_{r(v)}^* \leq ect_{Leaves(r(v))}^{LB3}$. Let us call $\Omega^{LB3} \subseteq Leaves(v)$ the optimal set to compute $ect_{Leaves(v)}^{LB3}$. Either:

- $ect_v^* = ect_{r(v)}^*$. This rule assumes $\Omega^{LB3} \subseteq Leaves(r(v))$. If this is the case, then we already know by induction that $ect_{r(v)}^* \le ect_{Leaves(r(v))}^{LB3}$.
- $ect_v^* = ect_{l(v)}^* + p_{r(v)} + \underline{tt}\big(|F_{r(v)} \setminus F_{l(v)}| + eI\big(F_{l(v)}, F_{r(v)}\big)\big)$. This rule assumes $\Omega^{LB3} \cap Leaves(l(v)) \ne \emptyset$. If this is the case, then one only needs to ensure that:

$$ect_{Leaves(l(v))}^{LB3} + p_{r(v)} + \underline{tt}\big(|F_{r(v)} \setminus F_{l(v)}| + eI\big(F_{l(v)}, F_{r(v)}\big)\big) \le ect_{Leaves(v)}^{LB3}$$

Intuitively, we only add to $ect_{Leaves(l(v))}^{LB3}$ a time quantity that was not considered in $ect_{Leaves(l(v))}^{LB3}$ and that has yet to be spent: durations of activities in $Leaves(r(v))$ and a number of transitions in $F_{r(v)} \setminus F_{l(v)}$ (plus an extra transition when the intersection between $F_{l(v)}$ and $F_{r(v)}$ is empty).

$\square$

Complexity. We use bit sets to represent the set of families in each node. The space complexity of the Θ-tree is therefore $\mathcal{O}(n|\mathcal{F}|)$. The set operations we use are *union, intersection, difference* and *cardinality*. Using bit sets, the 3 former ones are $\mathcal{O}(1)$ and the latter one is $\mathcal{O}(\log(|\mathcal{F}|))$ with a *binary population count* [18]. The time complexity of insertion and deletion of an activity in the Θ-tree is therefore $\mathcal{O}(\log(n)\log(|\mathcal{F}|))$.

Example 3. *Let us consider the activities presented in Fig. 2 (left). The transition matrix $\mathcal{TT}^{\mathcal{F}}$ between families is given in Fig. 2 (center). The pre-computed values of $\underline{tt}(k)$ are reported in Fig. 2 (right). Figure 3 illustrates the extended Θ-tree when all activities are inserted. Note that the value at the root of the tree is indeed a lower bound as the real ect is 85 and $ect_\Theta^{LB3} = 80$.*

	A_1	A_3	A_2	A_4
est	0	15	25	30
p	10	10	20	25
F	F_1	F_2	F_3	F_3

$$\mathcal{TT}^{\mathcal{F}} = \begin{pmatrix} 0 & 10 & 15 \\ 5 & 0 & 10 \\ 5 & 15 & 0 \end{pmatrix}$$

$\underline{tt}(k)$	k
0	0
1	0
2	5
3	15

Fig. 2. Example: four activities and their families (left), transition times for the families (center), and pre-computed lower bounds for the transition times (right).

3.3   Extending the Θ-Λ-tree with Families

The Edge-Finding (EF) algorithm requires an extension of the original Θ-tree, called Θ-Λ-tree [16]. In this extension, leaves are marked as either *white* or *gray*. White leaves represent activities in the set Θ and gray leaves represent activities that are in a second set, Λ, with $\Lambda \cap \Theta = \emptyset$. In addition to ect_v, a lower bound
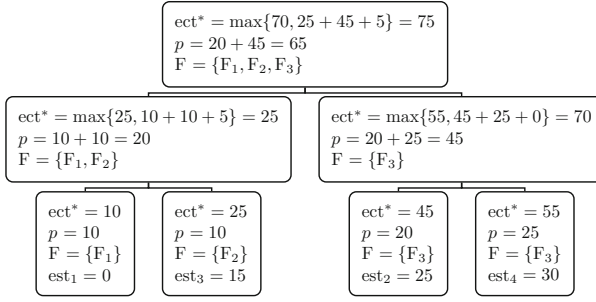
Fig. 3. A Θ-tree when all activities of Fig. 2 are inserted.

to the *ect* of Θ, a Θ-Λ-tree also aims at computing $\overline{ect}_v$, which is a lower bound to $\overline{ect}_{(\Theta,\Lambda)}$, the largest *ect* obtained by including *one* activity from Λ into Θ:

$$\overline{ect}_{(\Theta,\Lambda)} = \max_{A_i \in \Lambda} \ ect_{\Theta \cup \{A_i\}}$$

In addition to p_v, ect_v, the original Θ-Λ-tree structure also maintains $\overline{p}_v$ and $\overline{ect}_v$, respectively corresponding to p_v and ect_v, *if* a single gray activity in the sub-tree rooted at v maximizing $ect_{Leaves(v) \cup \{A_i\}}$ *was* included.

Our extension to the Θ-Λ-tree is similar to the one outlined in Sect. 3.2: in addition to the previous values, each internal node also stores F_v and $\overline{F}_v$ in order to compute the lower bounds ect_v^* and $\overline{ect}_v^*$. This latter value is defined as:

$$\overline{ect}^*_{(\Theta,\Lambda)} = \max \left\{ ect_\Theta^*, \ \max_{A_i \in \Lambda} \left\{ ect_{\Theta \cup \{A_i\}}^* \right\} \right\}$$

Adapting the rules for the Θ-Λ-tree requires caution when families are involved. In [5,16], the rules only use implicitly the information about *which* gray activity is considered in the update. In our case, the rules must consider explicitly where the used gray activity is located: either in the left subtree, denoted (L), or in the right subtree, denoted (R). The rules are then defined as:

$$\overline{ect}^*_v = \max \begin{cases} \overline{ect}^*_{l(v)} + p_{r(v)} + \underline{tt}\left(|F_{r(v)} \setminus \overline{F}_{l(v)}| + eI\left(\overline{F}_{l(v)}, F_{r(v)}\right)\right) & \text{(L)} \\ ect^*_{l(v)} + \overline{p}_{r(v)} + \underline{tt}\left(|\overline{F}_{r(v)} \setminus F_{l(v)}| + eI\left(F_{l(v)}, \overline{F}_{r(v)}\right)\right) & \text{(R)} \\ \overline{ect}^*_{r(v)} & \text{(R)} \end{cases}$$

$$\overline{F}_v = \begin{cases} \overline{F}_{l(v)} \cup F_{r(v)} & \text{(L)} \\ F_{l(v)} \cup \overline{F}_{r(v)} & \text{(R)} \end{cases}$$

$$\overline{p}_v = \begin{cases} \overline{p}_{l(v)} + p_{r(v)} & \text{(L)} \\ p_{l(v)} + \overline{p}_{r(v)} & \text{(R)} \end{cases}$$

In the rules above, the choice of which formula to use for $\overline{F}_v$ and $\overline{p}_v$ depends on the letter, either (L) or (R), associated with the term maximizing $\overline{ect}_v^*$, hence this value must be computed first. If a leaf v represents an activity A_i, then we

simply have $\overline{ect_v^*} = ect_i$, $\overline{p}_v = \mathrm{p}_i$, and $\overline{F}_v = \{F(A_i)\}$. The rules for p_v, ect_v, and F_v are as presented in Sect. 3.2, but one must also define, for a gray leaf v, $ect_v^* = -\infty$, $p_v = 0$, and $F_v = \emptyset$.

For space reasons, we do not present the proof of correctness of our recursive rules. As for the extended Θ-tree introduced in Sect. 3.2, the time complexity for the insertion and the deletion of an activity is $\mathcal{O}(\log(n)\log(|\mathcal{F}|))$.

4   Related Work

As described in a recent survey [1], scheduling problems with transition times can be classified in different categories. First the activities can be in *batch* (i.e. a machine allows several activities of the same batch to be processed simultaneously) or not. Transition times may exist between successive batches. A CP approach for batch problems with transition times is described in [16]. Secondly the transition times may be *sequence-dependent* or *sequence-independent*. Transition times are said to be sequence-dependent if their durations depend on both activities between which they occur. On the other hand, transition times are sequence-independent if their durations only depend on the activity after which they take place. The problem category we study in this article is non-batch sequence-dependent transition times problems.

Over the years, many CP approaches have been developed to solve such problems [2,5,7,10,19]. For instance, in [2], a Traveling Salesman Problem with Time Window (TSPTW) relaxation is associated to each resource. The activities used by a resource are represented as vertices in a graph, and edges between vertices are weighted with the corresponding transition times. The TSPTW obtained by adding time windows to vertices from bounds of corresponding activities is then resolved. If one of the TSPTW is found unsatisfiable, then the corresponding node of the search tree is pruned. A similar technique is used in [3] with additional propagators, which are, to the best of our knowledge, the state of the art propagators when families of activities are present.

State-of-the-art Filtering with Families

An idea from [17] that is also used in [3] is to pre-compute the exact minimal total transition time for every subset of families. For a subset of families $\mathcal{F}' \subseteq \mathcal{F}$, let $tt\,(\mathcal{F}')$ denote the minimal total transition time used for any activity set Ω such that $F_\Omega = \mathcal{F}'$. Similarly $tt\,(F_i \rightarrow \mathcal{F}')$ is the minimal total transition time when the processing starts with some activity of type $F_i \in \mathcal{F}'$, and $tt\,(\mathcal{F}' \rightarrow F_i)$ when it completes with an activity of type $F_i \in \mathcal{F}'$. We can pre-compute these values for every set of families $\mathcal{F}' \subseteq \mathcal{F}$ and every family $F_i \in \mathcal{F}'$ with a dynamic program running in $\Theta(|\mathcal{F}|^2.2^{|\mathcal{F}|})$ and requiring $\Theta(|\mathcal{F}|.2^{|\mathcal{F}|})$ of memory. For instance, for $tt\,(F_i \rightarrow \mathcal{F}')$, one defines:

$$\begin{cases} tt\,(F_i \rightarrow \{F_i\}) = 0 & \forall F_i \in \mathcal{F} \\ tt\,(F_i \rightarrow \{\mathcal{F}' \cup F_i\}) = \min_{F_j \in \mathcal{F}'} \{tt^{\mathcal{F}}_{F_i,F_j} + tt\,(F_j \rightarrow \mathcal{F}')\} & \forall \mathcal{F}' \subseteq \mathcal{F}, \forall F_i \in \mathcal{F} \setminus \mathcal{F}' \end{cases}$$

Based on these pre-computed values, which are assumed to be obtainable in $\mathcal{O}(1)$ once the pre-computation is made, two propagators are introduced in [3]:

– A DP-like propagator called UPDATEEARLIESTSTART running in $\mathcal{O}(n^2 \log(n))$.
– An EF-like propagator called PRIMALEDGEFINDING running in $\mathcal{O}(|\mathcal{F}|n^2)$.

Although the filtering obtained with these propagators can be stronger than their counterpart from [16] and our extensions, the time complexity of the propagators is quite high as compared to $O(n \log(n) \log(|\mathcal{F}|))$. In addition, they do not make use of a Not-First/Not-Last rule and the pre-computation of the minimal exact transition times for every subset of family is only tractable for small (typically less than 10) values of $|\mathcal{F}|$.

5   Experimentations

The experiments were conducted on JSPSDTT instances. We used AMD Opteron processors (2.7 GHz), the Java Runtime Environment 8 and the constraint solver *OscaR* [12]. The memory consumption was limited to 4 Gb.

Problem Instances. We have used two sets of instances. First, we used the standard **t2ps** instances from Brucker and Thiele [4]. However, there are only 15 of them, and we wanted to evaluate instances with more families, jobs, and machines in order to challenge the scalability of the different approaches. We therefore generated a new set of 315 instances, here referred to as **uttf**, with up to 50 jobs, 15 machines and 30 families.[2]

Compared Propagators. We compare models with the following propagators for Eq. (1):

– *binary-decomp*: binary decomposition of Eq. (1) only.
– *utt-no-families*: propagators for URTT from [5].
– *artigues-exact-tsp*: propagators of [3] using exact values for $tt(\mathcal{F})$, $tt(F \to \mathcal{F})$ and $tt(\mathcal{F} \to F)$.
– *artigues-lb-tsp*: propagators of [3] adapted to make use of cardinality-based lower bounds from [5] for $tt(\mathcal{F})$, $tt(F \to \mathcal{F})$ and $tt(\mathcal{F} \to F)$.
– *utt-families-exact-tsp*: propagators introduced in this paper making use of the exact values for $\underline{tt}(|\mathcal{F}|)$ computed with $\min_{\mathcal{F}':|\mathcal{F}'|=|\mathcal{F}|} tt(\mathcal{F}')$.
– *utt-families-lb-tsp*: propagators introduced in this paper making use of lower bounds for $\underline{tt}(|\mathcal{F}|)$. The bounds are computed with the lower bounds of [5].

All approaches also use the binary decomposition of Eq. (1) in order to ensure correctness as specialized propagators are generally not checking.

---

[2] The instances are available at http://becool.info.ucl.ac.be/resources/uttf-instances.

Replay Evaluation. In order to derive fair and representative conclusions about the propagators only (i.e., by removing the effects of the search heuristic), we used the *Replay* evaluation methodology [14]. First, for each instance, a *baseline* model is used to generate a search tree. This baseline model is, among the different compared approaches, the one that prunes the less the domains (here *binary-decomp*). Once the search tree is generated, it is *replayed* separately with each model. A replay basically consists in reapplying the exact same sequence of modifications to the constraint store (e.g., the branching constraints) that were used to generate the search tree with the baseline model.

The performance of those replays is then used to construct so-called *performance profiles* [6], that we built with a public web tool [15] made available to the community.[3] Performance profiles are cumulative distribution functions of a performance metric ratio τ. In our case, τ is a ratio of either time or number of backtracks. In the case of time, the function is defined as:

$$F_m(\tau) = \frac{1}{|\mathcal{I}|} \left| \left\{ i \in \mathcal{I} : \frac{time_{replay}(m, i)}{\min\limits_{m' \in M} time_{replay}(m', i)} \leq \tau \right\} \right| \tag{8}$$

where $\mathcal{I}$ is the set of considered instances, m is a model and M is the set of all models. The function is similar for the number of backtracks.

To generate the search tree, the Conflict Ordering Search [8] was used, as it was shown to be a good search strategy for scheduling problems. The generation lasted for 300 s, and we enforced a timeout of 1,800 s for the replay. If a timeout occurs for a model m, we consider that $\frac{time_{replay}(m,i)}{\min\limits_{m' \in M} time_{replay}(m',i)} = +\infty$. The running times reported here do not take into account the pre-computation step since they are negligible (generally less than 2 s and max 10 s).

Results on the t2ps Instances. Figures 4 and 5 provide the performance profiles for the time and number of backtracks, respectively. Figure 5 shows that, interestingly, *utt-families-lb-tsp* prunes exactly as much as *utt-families-exact-tsp*. This is due to the fact that our lower bounds are here able to compute the same values than $\min_{\mathcal{F}':|\mathcal{F}'|=|\mathcal{F}|} tt\,(\mathcal{F}')$. This suggests that we often do not have to compute the exact values for $tt\,(\mathcal{F})$ with the resource-consuming dynamic program, which is interesting since it is not tractable when there are many families. We can see that from a time perspective, our approach is the fastest for 80 % of the instances (*utt-families-exact-tsp* being here equivalent to *utt-families-lb-tsp*, see the function in $\tau = 1$ in Fig. 4). But our approach is also robust, as the other instances (i.e., the remaining 20 %) are solved within a factor $\tau < 2$ compared to the best model for those remaining instances. Considering the number of backtracks, our approach generally achieves less pruning than *artigues-exact-tsp* (not more than three times), but substantially more than *utt-no-families*. This lack of pruning as compared to *artigues-exact-tsp* is compensated in practice by the low time complexity. Although not reported, we tried

---

[3] Accessible at http://sites.uclouvain.be/performance-profile/.
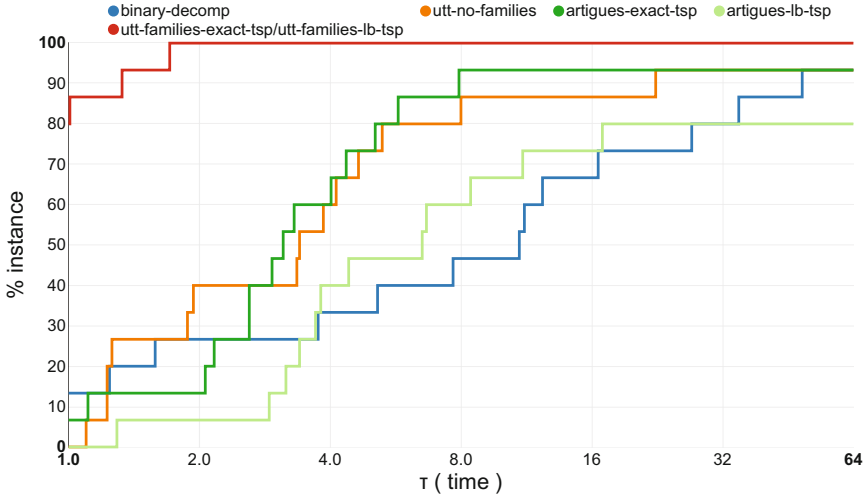
Fig. 4. Performance profiles on **t2ps** instances for the time metric.

to combine *utt-families-exact-tsp* and *artigues-exact-tsp* and the performances were close to the ones of *artigues-exact-tsp* alone, thus only inducing a small overhead when *utt-families-exact-tsp* does not provide additional pruning.

Results on the uttf Instances. First of all, we consider the approaches *artigues-exact-tsp* and *utt-families-exact-tsp* unable to solve (i.e., times out by default) the 120 instances (out of 315) with 20 families or more, since the pre-computation becomes too expensive in terms of CPU and memory usage according to our 4 Gb limitation.

Figures 6 and 7 provide the time performance profiles for the instances with strictly less than and with more than 20 families, respectively. Figure 6 shows that our approach still outperforms the other ones, even if it is the fastest on a smaller percentage of instances than for the **t2ps** instances. The instances being less structured, the gain in pruning is weaker as compared to the decomposition. However, our method catches up very quickly; for example, it is at most ∼ 1.3 and 2 times slower than the best approach for almost 60 % and 80 % of the instances, respectively. Another interesting point is that *utt-families-exact-tsp* and *utt-families-lb-tsp* have very similar time performances, while the values for $\underline{tt}(k)$ were here generally different (not reported here). This means that computing the exact values for $tt(\mathcal{F})$ is not mandatory[4] when used with our propagators, which is profitable since we also target scalability in terms of number of families.

Regarding the instances with more than 20 families (Fig. 7), our approach is significantly better than the other ones, as we are the fastest on almost 70 % of the instances and it is at most 4 times slower than the best approach on the

---

[4] Still, if it is available at a low cost, it can be beneficial to use it.
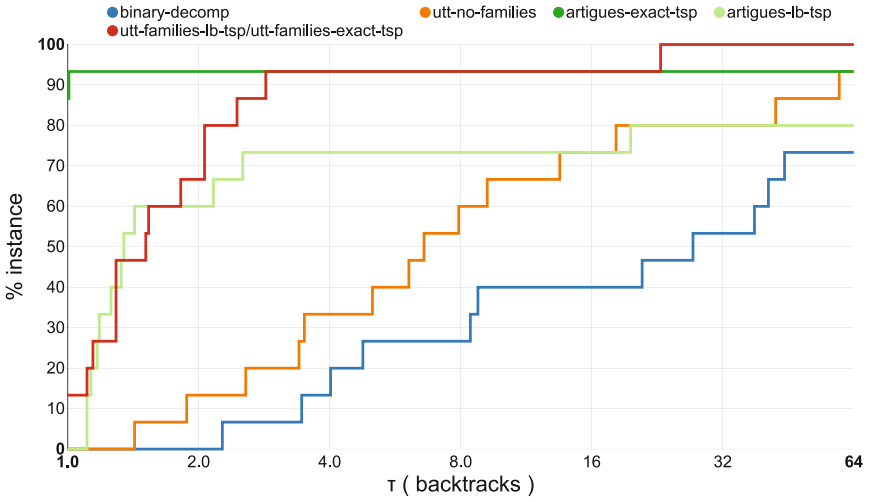
Fig. 5. Performance profiles on **t2ps** instances for the backtracks metric.
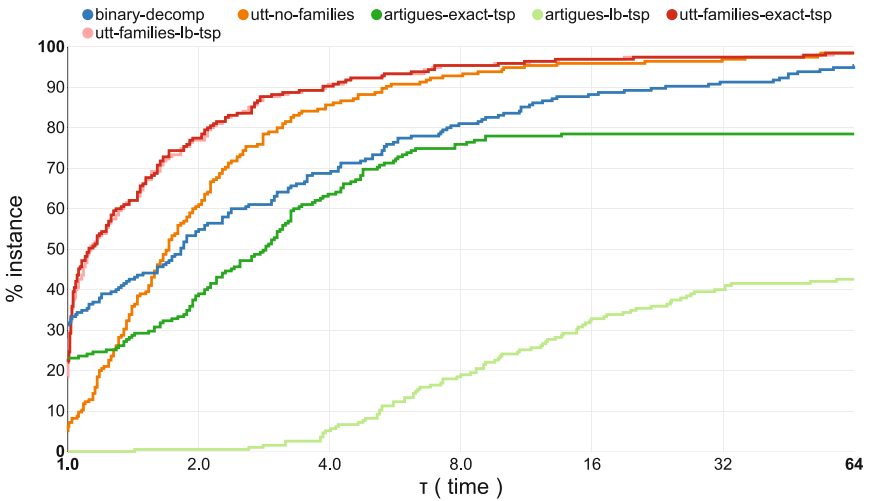
Fig. 6. Performance profiles on **uttf** instances with strictly less than 20 families for the time metric.

remaining instances. This teaches us that when more families are involved, our approach is both efficient and robust.

*Improvements on Open **t2ps** Instances.* Although not the focus of this paper, we were able to find tighter upper bounds for 3 of the 6 open **t2ps** instances within less than 5 min of computation. We simply combined our lightweight propagators with a LNS [13]. We used a basic relaxation of precedences of activities on the same machine combined with a *Set Times* [11] search strategy. The improved bounds are given in Table 1.
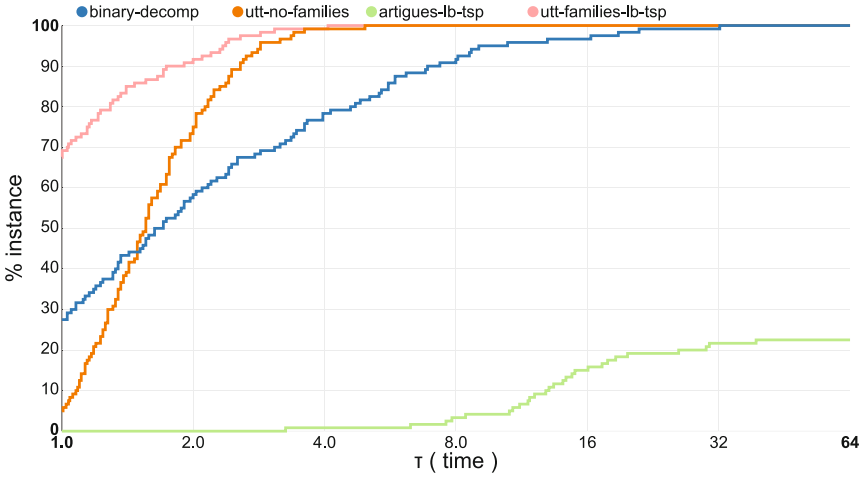
Fig. 7. Performance profiles on **uttf** instances with more than 20 families for the time metric.

Table 1. New upper bounds for open **t2ps** instances.

Upper bound	t2-ps11	t2-ps12	t2-ps15
Former	1,470	1,305	1,527
New	1,441	1,299	1,505

6   Conclusion

This paper has extended the algorithms and data structures for the unary resource, taking into account family-based transition times in order to perform additional propagation. The original data structures and algorithms have been adapted accordingly. The approach is therefore lightweight from both the time and space perspectives. Experiments conducted on the JSPSDTT have demonstrated that the introduced approach provides a substantial gain and is quite robust to changes in instance characteristics (e.g., number of activities and families).

Future work. We would like to consider other types of problems and combine this work with the use of good lower bounds in a branch-and-bound setting. More importantly, when there are no families defined *a priori* in an instance, we want to study the benefit of first creating them by means of *clustering* algorithms and then using the filtering introduced in this paper. This approach might prove to be helpful when the intra-cluster transition times are significantly smaller than the inter-cluster ones.

References

1. Allahverdi, A., Ng, C., Cheng, T.E., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. Eur. J. Oper. Res. **187**(3), 985–1032 (2008)
2. Artigues, C., Belmokhtar, S., Feillet, D.: A new exact solution algorithm for the job shop problem with sequence-dependent setup times. In: Régin, J.-C., Rueher, M. (eds.) CPAIOR 2004. LNCS, vol. 3011, pp. 37–49. Springer, Heidelberg (2004)
3. Artigues, C., Feillet, D.: A branch and bound method for the job-shop problem with sequence-dependent setup times. Ann. Oper. Res. **159**(1), 135–159 (2008)
4. Brucker, P., Thiele, O.: A branch & bound method for the general-shop problem with sequence dependent setup-times. Operations-Research-Spektrum **18**(3), 145–161 (1996)
5. Dejemeppe, C., Van Cauwelaert, S., Schaus, P.: The unary resource with transition times. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 89–104. Springer, Heidelberg (2015)
6. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
7. Focacci, F., Laborie, P., Nuijten, W.: Solving scheduling problems with setup times and alternative resources. In: AIPS, pp. 92–101 (2000)
8. Gay, S., Hartert, R., Lecoutre, C., Schaus, P.: Conflict ordering search for scheduling problems. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 140–148. Springer, Heidelberg (2015)
9. Gay, S., Schaus, P., De Smedt, V.: Continuous casting scheduling with constraint programming. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 831–845. Springer, Heidelberg (2014)
10. Grimes, D., Hebrard, E.: Job shop scheduling with setup times and maximal time-lags: a simple constraint programming approach. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 147–161. Springer, Heidelberg (2010)
11. Le Pape, C., Couronné, P., Vergamini, D., Gosselin, V.: Time-versus-capacity compromises in project scheduling (1994)
12. OscaR Team: OscaR: Scala in OR (2012). https://bitbucket.org/oscarlib/oscar
13. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
14. Van Cauwelaert, S., Lombardi, M., Schaus, P.: Understanding the potential of propagators. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 427–436. Springer, Heidelberg (2015)
15. Van Cauwelaert, S., Lombardi, M., Schaus, P.: A visual web tool to perform what-if analysis of optimization approaches. Technical report, UCLouvain (2016)
16. Vilím, P.: Global constraints in scheduling. Ph.D. thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, KTIML MFF, Universita Karlova, Malostranské námestı 2/25, 118 00 Praha 1, Czech Republic (2007)
17. Vilím, P., Barták, R.: Filtering algorithms for batch processing with sequence dependent setup times. In: Proceedings of the 6th International Conference on AI Planning and Scheduling, AIPS (2012)
18. Warren, H.S.: Hacker's Delight. Pearson Education, Upper Saddle River (2013)
19. Wolf, A.: Constraint-based task scheduling with sequence dependent setup times, time windows and breaks. GI Jahrestagung **154**, 3205–3219 (2009)
20. Zampelli, S., Vergados, Y., Van Schaeren, R., Dullaert, W., Raa, B.: The berth allocation and quay crane assignment problem using a CP approach. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 880–896. Springer, Heidelberg (2013)

Application Track

A Constraint Programming Approach to Multi-Robot Task Allocation and Scheduling in Retirement Homes

Kyle E.C. Booth[(✉)], Goldie Nejat, and J. Christopher Beck

Department of Mechanical and Industrial Engineering,
University of Toronto, Toronto, ON M5S 3G8, Canada
{kbooth,nejat,jcb}@mie.utoronto.ca

Abstract. We study the application of constraint programming (CP) to the planning and scheduling of multiple social robots interacting with residents in a retirement home. The robots autonomously organize and facilitate group and individual activities among residents. The application is a multi-robot task allocation and scheduling problem in which task plans must be determined that integrate with resident schedules. The problem involves reasoning about disjoint time windows, inter-schedule task dependencies, user and robot travel times, as well as robot energy levels. We propose mixed-integer programming (MIP) and CP approaches for this problem and investigate methods for improving our initial CP approach using symmetry breaking, variable ordering heuristics, and large neighbourhood search. We introduce a relaxed CP model for determining provable bounds on solution quality. Experiments indicate substantial superiority of the initial CP approach over MIP, and subsequent significant improvements in the CP approach through our manipulations. This work is one of the few, of which we are aware, that applies CP to multi-robot task allocation and scheduling problems. Our results demonstrate the promise of CP scheduling technology as a general optimization infrastructure for such problems.

1 Introduction

The progressive aging of populations, as observed primarily within developed countries, has important implications in a number of societal areas, including health and social care services for the elderly [1]. Such demographic trends have resulted in a dramatic increase in the number of seniors residing in retirement and nursing homes [2]. This increase in demand for care services, combined with a reduction in the working age population, will inevitably result in greater pressures on the quality of elderly care infrastructure, risking deterioration in the provision of medical services, daily assistance, social interaction, and overall quality of life for residents. Due to these demographic and industry dynamics, the investigation of the role of autonomous robotics within healthcare has been discussed for a number of decades, though primarily with respect to robots assisting physical rehabilitation. The design and deployment of socially assistive

robots for retirement home applications and elderly care is a more recent development [3]. Such social robots alleviate workforce pressures associated with the daily operation of retirement homes and work to give assistance through the autonomous facilitation of cognitively and socially stimulating leisure activities.

In this paper, we contribute a novel application of constraint programming (CP) to the automated planning and scheduling of a team of social robots in a retirement home. Our larger project involves the robots providing social and cognitive stimulation through the facilitation of bingo games involving multiple residents and telepresence sessions between residents and their family members. Here, we propose CP as part of a task planning system that must autonomously allocate, schedule, and facilitate these single and multi-resident leisure activities throughout the course of the day, while adhering to daily resident calendars defining their availability. The problem involves reasoning about which tasks should be implemented (i.e. *planning*), as well as which robot should facilitate each task and at what time (i.e. *scheduling*).

In the field of robotics, multi-robot task allocation (MRTA) aims to solve robot coordination problems pertaining to task decomposition from high-level goals, task distribution, and task scheduling. We extend the previously proposed single robot version of the retirement home problem [4,5], to an MRTA problem. We investigate mixed-integer programming (MIP) and CP as allocation and scheduling strategies. These approaches model disjoint time windows, robot and user travel times within the retirement home, inter-schedule task dependencies, and robot energy consumption/replenishment. We investigate enhancements of our initial CP approach through grouped variable ordering heuristics and large neighbourhood search (LNS), and present a relaxed CP formulation used for determining provable bounds on solution quality. Numerical results indicate substantial superiority of the CP formulation over MIP, and we show significant improvements of the CP approach through our manipulations of the search. This experimentation illustrates CP scheduling technology as a promising general optimization framework for MRTA problems.

2 Related Work

CP has been applied to a wide range of combinatorial optimization problems, excelling most notably in scheduling applications [6], where it has established itself as a strong competitor to mathematical programming-based approaches, often out-performing state-of-the-art MIP solvers [7]. The flexible nature of CP, combined with its proficiency at representing and solving particular combinatorial substructure (e.g. problems with task sequencing) has led to its integration with other methods, producing stronger hybrid approaches. Examples of this integration include logic-based Benders decomposition (LBBD) [8] and constraint-integer programming (CIP) [9]. CP has also been used in combination with Local Search (LS) in the Large Neighbourhood Search (LNS) [10] framework. Indeed, commercial CP software has benefited tremendously from this integration as seen within the incorporation of self-adapting LNS in state-of-the-art constraint solvers [11].

Initial approaches to MRTA problems used dispatch-style methods where a single task was allocated and executed before the next allocation was made [12,13]. More recent approaches utilize decentralized methods such as market-based strategies [14], auction-based approaches [15], and distributed local task-swapping [16]. In the past decade, efforts have been made to use linear and integer programming techniques [5,17], largely due to attractive bounds on solution quality. CP has been proposed as a suitable candidate approach for these problems [18,19], however, the application of CP to multi-robot task planning and scheduling is, to the best of our knowledge, limited in the literature. The MAC-BETH [20] architecture makes use of a combination of hierarchical task networks and CP, where a human user specifies missions to a team of autonomous agents via a playbook graphic user interface. Another proposed method uses distributed constraint satisfaction problems (disCSP) to solve multi-robot exploration problems [21].

Socially assistive robots for elderly care have seen growing attention within the literature [22]. For the retirement home application studied in this paper, existing related work has presented temporal planning, MIP, and CP approaches for solving the single-robot task planning variant of the problem [4,5], where CP was demonstrated to outperform the other techniques. Multiple robot scenarios have also been recently studied [23], where a planning and scheduling architecture was introduced using off-the-shelf temporal planners for a specialization of the problem studied in this paper.

3   Problem Definition

Given a set of robots, R, a set of possibly optional tasks, T, and a problem-specific cost function, our MRTA problem involves determining a mapping of tasks to robots, $f : T \rightarrow R$, as well as an assignment of start times to tasks, such that the objective is optimized. In this section we discuss specific problem parameters and objectives associated with our retirement home application.

3.1   Parameters

We consider a set of users (retirement home residents), $U := \{u_1, u_2, ..., u_n\}$, where each user, $u_i \in U$, has a unique daily calendar, $\Sigma_i := \{\sigma_{i1}, \sigma_{i2}, ..., \sigma_{i5}\}$, identifying five busy periods where the user is not available for interaction. For modeling purposes, we treat each busy period as a required task defined on a closed interval with a fixed start and end time. These intervals include one hour breaks for breakfast (8:00–9:00 AM), lunch (12:00–1:00 PM), and dinner (5:00–6:00 PM), as well as two additional breaks with duration ranging from 30 min to one hour. An estimate of user movement speed, ν_u in metres/minute, is used to approximate travel times between locations.

We also consider a set of robots, $R := \{r_1, r_2, ..., r_m\}$, that facilitate human-robot interaction (HRI) tasks within the retirement home. Each robot, $r_k \in R$, starts and ends in the *robot depot*, a location containing a recharging station.

Robot movement speed, ν_r in metres/minute, is known, as well as lower and upper limits on battery level, β_{min} and β_{max}, respectively. Robot energy consumption rates are defined for robot movement, ξ_Δ, and consumption (or replenishment, with a negative rate, for recharge tasks) for task j as ξ_j.

The retirement home contains a set of locations, $L := \{\ell_r, \ell_g, \ell_\sigma\} \cup \{\ell_1, \ell_2, ..., \ell_n\}$, representing the robot depot, games room, meal/break room, and a personal room for each of the users. Distances between any pair of locations ℓ_a and ℓ_b are known, and defined as $\delta_{(a,b)}$, in metres. Travel time matrices are generated for users, $\Delta^u := \{\frac{\delta_{(a,b)}}{\nu_u} : (a,b) \in L \times L\}$, and for robots, $\Delta^r := \{\frac{\delta_{(a,b)}}{\nu_r} : (a,b) \in L \times L\}$, where travel times are estimated in minutes.

The problem considers individual and group HRI tasks, each requiring a single robot facilitator. These task types are: *telepresence* tasks (individual), *bingo games* (group), and *bingo game reminders* (individual). The set of telepresence tasks is defined as $P := \{p_1, p_2, ..., p_n\}$, where each task corresponds to a user; these tasks take place in the personal room of the user and have a duration of 30 min. The set of bingo game tasks, $G := \{g_1, g_2, ..., g_{UB_1}\}$, is defined based on a calculated upper bound, UB_1, as the number of games that can be facilitated is unknown a priori. These tasks take place in the games room and have a duration of 60 min. The necessity for this upper bound illustrates an important limitation when using scheduling methods for problems with underlying planning characteristics: a predefined number of tasks is required. We define the set of available reminder tasks as $\mathcal{M} := \bigcup_{i=1}^{n} M_i$ where the subset of reminder tasks for each user, $u_i \in U$, is defined as $M_i := \{m_{i1}, m_{i2}, ..., m_{iUB_1}\}$. Each reminder takes place in the personal room of the participating user and has a duration of two minutes. The duration of busy period, telepresence, bingo game, and reminder tasks are represented as d_j for task j. We also define a set of available *recharge* tasks, $\mathcal{C} := \bigcup_{k=1}^{m} C_k$, where the subset of these tasks for each robot, $r_k \in R$, is defined as $C_k := \{c_{k1}, c_{k2}, ..., c_{kUB_2}\}$. The number of these tasks available for each robot is defined based on the calculated upper bound, UB_2, as the number of recharge tasks a robot may require is also unknown a priori. The duration (in minutes) of each recharge task varies within the closed interval $[0, \frac{\beta_{max} - \beta_{min}}{(-1) \cdot \xi_j}]$ for $j \in C_k$, $\forall r_k \in R$.

We define the set of all tasks potentially involving each user, $u_i \in U$, as $T_i^u := \{\Sigma_i \cup p_i \cup G \cup M_i\}$, and mandatory start and end dummy tasks with zero duration for users as $\dot{u}$ and $\ddot{u}$, respectively. These tasks facilitate user task sequencing and have zero transition time to all other task locations. We define the set of all tasks potentially involving each robot, $r_k \in R$, as $T_k^r := \{P \cup G \cup \mathcal{M} \cup C_k\}$, with mandatory start and end dummy tasks with zero duration as $\dot{r}$ and $\ddot{r}$, respectively. These tasks are located at the robot depot, and have associated spatial transition times to other tasks, ultimately ensuring that robot schedules start and end at the robot depot.

3.2   Objective

Given a single day (7:00 AM–7:00 PM) planning horizon, H, a time-extended allocation of tasks to robots must be determined that integrate with user sched-

ules. The battery level of each robot, $r_k \in R$, is known throughout the day, and must stay within the specified closed interval, $[\beta_{min}, \beta_{max}]$. Each user must participate in exactly one telepresence activity but user participation in bingo games is optional. If a user participates in a bingo game activity, the associated reminder task must be done before the game. The optimization objective of the problem is to maximize bingo game participation over all users. Solutions with equivalent bingo game participation are prioritized by favoring schedules with fewer robot recharge tasks.

A feasible solution to a small problem instance is illustrated in Fig. 1. Telepresence (orange), bingo game (blue), reminder (grey), and recharge (yellow) tasks are all represented, as well as user busy periods (green). We note that unless tasks occur in the same location (e.g. reminder and telepresence in a user personal room), the tasks are never scheduled immediately next to each other. This is due to the modeling of travel times for both robot and users.

Though in this particular problem definition we are generating a single time-extended plan, complete solution methods for this problem will need to incorporate replanning due to likely discrepancies during schedule execution (e.g. a missed telepresence). As such, we look to generate high-quality allocations within reasonably short time-frames ($\leq 5\,\mathrm{min}$).
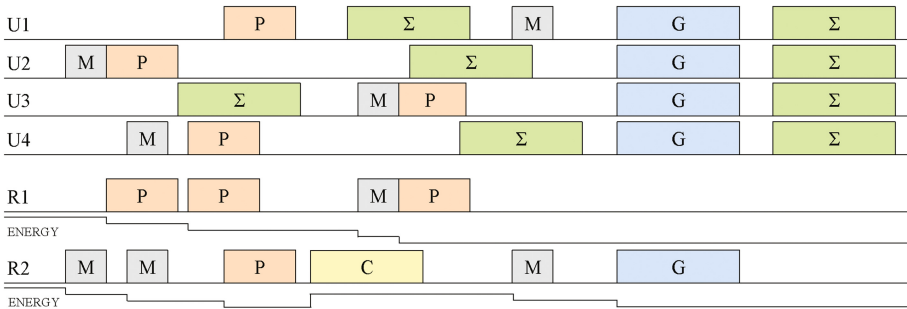
Fig. 1. Time-extended MRTA for a retirement home: feasible task allocations and schedules. Instance size: $|U| = 4, |R| = 2$. (Color figure online)

4 Task Allocation and Scheduling Models

In this section, we present task allocation and scheduling formulations using CP and MIP. We formally define both models and discuss key modeling considerations made.

4.1 Constraint Programming Model

We present a CP model in Fig. 2. We utilize cumulative variables to model robot energy levels and *optional interval variables* to model tasks, which are decision

variables whose possible values are a convex interval: $\{\perp\}\cup\{[s,e)|s,e\in\mathbb{Z},s\leq e\}$, where s and e are the start and end values of the interval and $\perp$ is a special value indicating the variable is not present in the solution [24]. The length of each interval variable corresponds to the duration of its associated task. The variable $\texttt{Pres}(var)$ is 1 if interval variable var is present in the solution, and 0 otherwise. Model constraints are only enforced on interval variables that are present in the solution. $\texttt{Start}(var)$, $\texttt{End}(var)$, and $\texttt{Length}(var)$ return the start time, end time, and length, respectively, of the interval variable var.

We define the decision variables used in the CP formulation as follows:

$x_{ij} := (interval)$ present if user u_i attends task j and absent otherwise,
$y_{kj} := (interval)$ present if robot r_k facilitates task j and absent otherwise,
$E_k := (cumulative)$ energy level of robot r_k throughout the schedule.

$$max \quad \sum_{u_i\in U}\sum_{j\in G}\texttt{Pres}(x_{ij})-0.01\cdot\sum_{r_k\in R}\sum_{j\in C_k}\texttt{Pres}(y_{kj}) \tag{1}$$

$$s.t. \quad \texttt{NoOverlap}([x_{i\dot{u}},x_{i1},x_{i2},...,x_{i|T_i^u|},x_{i\ddot{u}}],\Delta^u), \quad \forall u_i\in U \tag{2}$$

$$\texttt{NoOverlap}([y_{k\dot{r}},y_{k1},y_{k2},...,y_{k|T_k^r|},y_{k\ddot{r}}],\Delta^r), \quad \forall r_k\in R \tag{3}$$

$$\texttt{Pres}(x_{ip_i})=\sum_{r_k\in R}\texttt{Pres}(y_{kp_i})=1, \quad \forall u_i\in U \tag{4}$$

$$\texttt{Pres}(x_{ij})\leq\sum_{r_k\in R}\texttt{Pres}(y_{kj})\leq 1, \quad \forall u_i\in U;j\in G \tag{5}$$

$$\texttt{End}(x_{im_{ij}})\leq\texttt{Start}(x_{ij}), \quad \forall u_i\in U;j\in G \tag{6}$$

$$\sum_{r_k\in R}\texttt{Pres}(y_{km_{ij}})=\texttt{Pres}(x_{ij}), \quad \forall u_i\in U;j\in G \tag{7}$$

$$\texttt{StartAtStart}(x_{ij},y_{kj},0), \quad \forall u_i\in U;r_k\in R;j\in T_i^u\cap T_k^r \tag{8}$$

$$E_k=\sum_{j\in T_k^r\cup\{\ddot{r}\}}\texttt{StepAtStart}(j,-\theta_{kj}), \quad \forall r_k\in R \tag{9}$$

$$\theta_{kj}=\texttt{Length}(y_{kj})\cdot\xi_j+\Delta_{(pre_j,j)}^r\cdot\xi_\Delta, \tag{10}$$

$$0\leq\texttt{Length}(y_{kj})\leq\frac{\beta_{max}-\beta_{min}}{(-1)\cdot\xi_j}, \quad \forall r_k\in R;j\in C_k \tag{11}$$

$$\beta_{min}\leq E_k\leq\beta_{max}, \quad \forall r_k\in R \tag{12}$$

$$\texttt{Pres}(x_{ij})=1,\texttt{Start}(x_{ij})=\sigma_{ij}, \quad \forall u_i\in U;j\in\Sigma_i \tag{13}$$

$$\texttt{Pres}(x_{ij})\in\{0,1\},\texttt{Start}(x_{ij})\in[0,H], \quad \forall u_i\in U;j\in T_i^u\setminus\Sigma_i \tag{14}$$

$$\texttt{Pres}(y_{kj})\in\{0,1\},\texttt{Start}(y_{kj})\in[0,H] \quad \forall r_k\in R;j\in T_k^r \tag{15}$$

Fig. 2. Constraint programming model

Objective (1) maximizes the total number of bingo games played across all users, prioritizing solutions with fewer recharge tasks in the event of equivalent group activity participation. Constraints (2) and (3) encapsulate the sequencing requirement for all potential tasks associated with each user and each robot, including required dummy tasks. We utilize the $\texttt{NoOverlap}$ global constraint

which performs efficient domain filtering on the interval variable start times by reasoning about task time windows, processing times, and the relationship that no pair of tasks can overlap in time, with consideration for transition times [6]. In our model, the `NoOverlap` constraints enforced on users differ from those enforced on robots due to the different tasks within the sets T_i^u and T_k^r. Since both users and robots move within the retirement home throughout the day, these constraints ensure that their final task plans are properly sequenced and allow for transition times through the inclusion of the Δ^u and Δ^r transition time matrices.

Constraint (4) ensures that, for each user, exactly one robot facilitates the required telepresence task. Constraint (5) links the user participation bingo game variables to the robot facilitation variables; the decision for user $u_i \in U$ to attend bingo game $j \in G$ is bounded by the presence of a robot $r_k \in R$ facilitating that game. This constraint also restricts each bingo game to be facilitated by at most one robot. Constraint (6) enforces the precedence relationship between user bingo game participation and reminder tasks. Constraint (7) ensures that if a user is attending a bingo game, he/she must receive exactly one reminder for that particular bingo game, and zero otherwise.

Constraint (8) ensures that the start times of intersecting tasks between user and robot schedules are synchronized. We use the `StartAtStart` constraint which ensures that, whenever both variables are present, the distance between their start times, $\texttt{Start}(x_{ij}) - \texttt{Start}(y_{kj}) = 0$. Constraints (9)–(12) represent the energy-related constraints for each robot. Collectively, these constraints ensure that the recharge task length and the cumulative function energy level variable, E_k, stay within specified bounds while using the `StepAtStart` global constraint to model the various energy consumptions of the tasks. The term θ_{kj} in Eq. (10) represents the energy consumption (or replenishment) of facilitating task j combined with the energy consumed in travelling to the location of the task from the previous location in the sequence. Constraint (13) identifies user busy period tasks as mandatory. Constraints (14) and (15) identify the optionality of the interval variables, as well as start time domains.

4.2   Mixed-Integer Programming Model

For comparison purposes, we also present a MIP model for the problem as defined in Fig. 3. The model is based on the formulation for the electric vehicle-routing problem with time windows (E-VRPTW) [25], treating each of the sets T_i^u and T_k^r as completely-connected graphs with edge-weights representing travel times between locations. We make extensive use of Miller-Tucker-Zemlin (MTZ) [26] sequencing constraints to model task start times and robot energy levels throughout the planning horizon. We extend the E-VRPTW by including task synchronization constraints, a concept that has been applied within the context of vehicle routing previously [27], as well as problem-specific inter-schedule task dependencies (i.e. reminders delivered before bingo games).

We define the decision variables used in the MIP formulation as follows:

$x_{ij} := (binary)$ 1 if user u_i attends task j and 0 otherwise,
$y_{kj} := (binary)$ 1 if robot r_k facilitates task j and 0 otherwise,
$\alpha_{ijl} := (binary)$ 1 if task j directly precedes task l for user u_i and 0 otherwise,
$\gamma_{kjl} := (binary)$ 1 if task j directly precedes task l for robot r_k and 0 otherwise,
$\phi_{ij} := (integer)$ start time of task j in the schedule of user u_i,
$\psi_{kj} := (integer)$ start time of task j in the schedule of robot r_k,
$D_{kj} := (integer)$ length of task j in the schedule of robot r_k,
$\epsilon_{kj} := (integer)$ energy level of robot r_k after completing task j.

$$max \quad \sum_{u_i \in U} \sum_{j \in G} x_{ij} - 0.01 \cdot \sum_{r_k \in R} \sum_{j \in C_k} y_{kj} \tag{16}$$

$$s.t. \quad \sum_{l \in T_i^u \cup \{\dot{u}\}, l \neq j} \alpha_{ijl} = x_{ij}, \qquad \forall u_i \in U; j \in T_i^u \cup \{\dot{u}\} \tag{17}$$

$$\sum_{j \in T_i^u \cup \{\dot{u}\}, j \neq l} \alpha_{ijl} = x_{il}, \qquad \forall u_i \in U; l \in T_i^u \cup \{\ddot{u}\} \tag{18}$$

$$\sum_{l \in T_k^r \cup \{\dot{r}\}, l \neq j} \gamma_{kjl} = y_{kj}, \qquad \forall r_k \in R; j \in T_k^r \cup \{\dot{r}\} \tag{19}$$

$$\sum_{j \in T_k^r \cup \{\dot{r}\}, j \neq l} \gamma_{kjl} = y_{kl}, \qquad \forall r_k \in R; l \in T_k^r \cup \{\ddot{r}\} \tag{20}$$

$$\phi_{ij} + d_j + \Delta_{(j,l)}^u \leq \phi_{il} + H \cdot (1 - \alpha_{ijl}), \qquad \forall u_i \in U; j, l \in T_i^u, j \neq l, \tag{21}$$

$$\psi_{kj} + D_{kj} + \Delta_{(j,l)}^r \leq \psi_{kl} + H \cdot (1 - \gamma_{kjl}), \qquad \forall r_k \in R; j, l \in T_k^r, j \neq l, \tag{22}$$

$$\epsilon_{kl} + D_{kj} \cdot \xi_j + \Delta_{(j,l)}^r \cdot \xi_\Delta \leq \epsilon_{kj} + \beta_{max} \cdot (1 - \gamma_{kjl}), \qquad \forall r_k \in R; j, l \in T_k^r, j \neq l \tag{23}$$

$$\phi_{ij} = \psi_{kj}, \qquad \forall u_i \in U; r_k \in R; j \in T_i^u \cap T_k^r, \tag{24}$$

$$x_{ip_i} = \sum_{r_k \in R} y_{kp_i} = 1, \qquad \forall u_i \in U \tag{25}$$

$$x_{ij} \leq \sum_{r_k \in R} y_{kj} \leq 1, \qquad \forall u_i \in U; j \in G \tag{26}$$

$$\sum_{r_k \in R} y_{km_{ij}} = x_{ij}, \qquad \forall u_i \in U; j \in G \tag{27}$$

$$D_{kj} = d_j, \qquad \forall r_k \in R; j \in T_k^r \setminus C_k \tag{28}$$

$$0 \leq D_{kj} \leq \frac{\beta_{max} - \beta_{min}}{(-1) \cdot \xi_j}, \qquad \forall r_k \in R; j \in C_k \tag{29}$$

$$\beta_{min} \leq \epsilon_{kj} \leq \beta_{max}, \qquad \forall r_k \in R; j \in T_k^r \cup \{\ddot{r}\} \tag{30}$$

$$x_{ij} = 1, \phi_{ij} = \sigma_{ij}, \qquad \forall u_i \in U; j \in \Sigma_i \tag{31}$$

$$x_{ij}, \alpha_{ijl} \in \{0, 1\}, \ \phi_{ij} \in [0, H], \qquad \forall u_i \in U; j, l \in T_i^u \setminus \Sigma_i \tag{32}$$

$$y_{kj}, \gamma_{kjl} \in \{0, 1\}, \ \psi_{kj} \in [0, H] \qquad \forall r_k \in R; j \in T_k^r \tag{33}$$

Fig. 3. Mixed-integer programming model

Objective (16) is functionally equivalent to the objective of the CP model. Constraints (17) and (18) represent the node degree constraints for user tasks, and Constraints (19) and (20) the node degree constraints for robot tasks. Constraints (21) and (22) are MTZ sequencing constraints used to determine valid

start times of user and robot tasks while adhering to task duration and transition times. Constraint (23) uses MTZ sequencing to model robot energy level, where energy is consumed or replenished depending on the task sequencing.

Constraint (24) synchronizes the start times of intersecting tasks between user and robot schedules, necessary for the integration of these task plans. Constraint (25) ensures that each user participates in exactly one telepresence task, and that each of these tasks is facilitated by exactly one robot. Constraint (26) constrains bingo game facilitation to at most one robot, and ensures user participation is bounded by this value. Constraint (27) ensures that if a user participates in a bingo game, he/she receives exactly one reminder facilitated by a single robot.

Constraint (28) defines the length of non-recharge robot tasks to be constant and Constraint (29) identifies the bounds on variable-length recharge tasks. Constraint (30) defines the acceptable bounds on robot battery level and the remainder of the model, Eqs. (31)–(33), dictates the domains of the decision variables as fixed, binary, or positive integer.

4.3   Modeling Considerations

The schedules produced for the users and robots must be temporally synchronized, must accurately model robot and user travel times, and must ensure adherence to the energy capacity of the robots. This section identifies key considerations made when modeling these complex relationships.

Schedule Synchronization. Task synchronization between user and robot schedules is a primary concern in this problem. While proposed methods for the single-robot retirement home application [5,28] have accounted for transition times between robot tasks, they have assumed users travel between locations instantly. As a result, if a candidate start time for a robot-facilitated task did not conflict with the availability calendar of that user, the start time was considered valid (assuming task duration and end time did not pose conflict).

When user movement within the environment is relaxed, straightforward modeling within CP would utilize the ForbidExtent(var, f) global constraint, which prevents an interval variable var from overlapping a time point t where $f(t)$, an integer step function, is equal to 0 [24]. This constraint represents a natural way to model relationships involving disjoint resource time windows or calendars, supplementing user-sequencing Constraint (2) within our formulation. This method of modeling is inaccurate as it does not represent the travel times of users to and from the locations of subsequent tasks nor break periods. To remedy this, we include NoOverlap global constraints for both users and robots.

Properly accounting for both user and robot movement within the environment brings further modeling challenges pertaining to schedule synchronization. Since user availability is now dependent on spatial transition times in addition to their calendars, the temporal synchronization of a task involving both a user and robot on their respective schedules is necessary. This requirement is achieved

within the CP and MIP models using Constraints (8) and (24), respectively. The modeling presented is then further strengthened by noting that tasks involving a one-to-one mapping of robots to users (e.g. telepresence and reminder tasks) can be simultaneously represented on both user and robot schedules. Tasks involving a one-to-many mapping of robots to users (e.g. bingo game activities) are linked with the aforementioned synchronization constraints.

Symmetry Breaking. The problem has a number of inherent symmetries due to the homogenous nature of resources and tasks. We investigate a number of symmetry breaking options in efforts to reduce the search. These constraints are formulated in CP, though similar MIP constraints can be expressed with binary variables. For a given robot, each of the recharging tasks available to it are identical. Our models, as formulated, treat a single recharge solution using recharge task, c_{kj}, as functionally different than a solution using recharge task, c_{kl}, even if the start times and durations of each are the same. These symmetries can be broken using the following CP constraint to order the use of recharge tasks:

$$\mathtt{Pres}(y_{kj+1}) \leq \mathtt{Pres}(y_{kj}), \forall r_k \in R, j \in C_k \setminus \{c_{kUB_2}\} \tag{34}$$

We can also consider breaking symmetries pertaining to bingo game tasks, as these tasks are also homogeneous. We break these symmetries by enforcing lexicographic ordering within robot facilitation decisions and user participation. We enforce an ordering on bingo game tasks with the following constraints:

$$\sum_{r_k \in R} \mathtt{Pres}(y_{kj+1}) \leq \sum_{r_k \in R} \mathtt{Pres}(y_{kj}), \forall j \in G \setminus \{g_{UB_1}\} \tag{35}$$

$$\sum_{u_i \in U} \mathtt{Pres}(x_{ij+1}) \leq \sum_{u_i \in U} \mathtt{Pres}(x_{ij}), \forall j \in G \setminus \{g_{UB_1}\} \tag{36}$$

Since bingo game tasks are linked among users and robots, symmetry breaking can only be expressed over the sum of such variables. As previously noted [29], symmetry breaking can be counterproductive and delay the discovery of feasible solutions. We investigate these constraints experimentally in Sect. 6.

5   CP Search Manipulations

As presented in Sect. 6, the initial MIP model exhibits very poor performance when compared to the initial CP approach. As such, we pursue CP as the more suitable technology for the given application. In this section we discuss methods for increasing the performance of the initial CP formulation, using grouped variable orderings heuristics and large neighbourhood search.

5.1   Grouped Variable Ordering Heuristics

One of the key focuses of this work is to determine if CP can be used to generate time-extended allocations within realistic timeframes ($\leq 5 \, \mathrm{min}$). In order to increase the performance of the CP formulation, we conduct a detailed investigation of *grouped variable ordering heuristics*, specific instantiation orderings of

groups of variables, to uncover elements of problem structure and help reduce the search space.

We define groups of variables and then instantiate each group according to a specified order within the search. Variable groups that appear earlier in the ordering have all of their elements instantiated before subsequent variables are considered. For the purposes of our investigation, we consider groups of variables associated with robot bingo game facilitation, bingo user participation, user telepresence participation, reminder participation, and robot recharge tasks. We implement instantiation orderings over variable groups defined as: $\mathcal{V} := \{\{y_{kg_1}, ..., y_{kg_{UB_1}}\}, \{x_{ig_1}, ..., x_{ig_{UB_1}}\}, \{x_{ip_1}, ..., x_{ip_n}\}, \{x_{im_{ig_1}}, ..., x_{im_{ig_{UB_1}}}\},$ $\{y_{kc_{k1}}, ..., y_{kc_{kUB_2}}\}\}$ for all robots $r_k \in R$ and users $u_i \in U$. Within these variable groups we investigate orderings on all possible subsets of $\mathcal{V}$ of size one and two (single and double stage). Problem variables not included in the selected subset will be instantiated after those selected. By inspecting Fig. 1, it is clear that instantiating the set of $\{y_{kg_1}, ..., y_{kg_{UB_1}}\}$ variables for all $r_k \in R$, as detailed in Constraint (5), will have high impact on other variables and thus may be promising candidates for early instantiation decisions.

For the first set of experiments, we consider subsets of $\mathcal{V}$ of size one resulting in five orderings. The remainder of the variables are then instantiated using the default solver strategy. The second set of experiments uses a double-stage search phase with subsets of size two. We explore all two-stage permutations of decision variable groups in $\mathcal{V}$, yielding 20 unique group orderings. With this two-stage assessment we hope to uncover findings pertaining to problem structure that may not be apparent upon initial inspection.

5.2 Large Neighbourhood Search

Large neighbourhood search (LNS) [10] is a method that combines local search (LS) with constraint programming (CP). It has proven to be effective for solving large, complex optimization problems [30]. We implement LNS in order to further improve solution quality on larger instances of our problem, using a variation of the *time window neighbourhood* selection heuristic [30]. Other selection heuristics that exploit problem-specific structure did not perform as well.

As initial solutions to the global problem are often of low quality, we allot one minute of run-time to a CP search using our best grouped variable ordering heuristic to find an incumbent solution. This initial search helps ensure that the LNS procedure begins with a high-quality solution, ultimately improving the performance of the method. Next, with variable set N, we unassign all variables that are: (i) present, and (ii) have start times within the current time window (initially this window is defined on the closed interval [7:00AM, 10:00AM]). We fix the remainder of the solution variable start times; the unassigned set is S, and the fixed set $r := N \setminus S$. We solve the resultant problem to try to quickly find improving solutions, if they exist. The time limit used here is set to 20 s, and a backtrack limit is enforced to prevent fruitless exploration. If the solution is improved, we reset the time window to its initial interval, replace the incumbent

solution and repeat. In the event the solution does not improve, we shift our time window to the right (i.e. later in time) by one hour and repeat the process. If all time windows, of the current size, are explored without improvement (with final window [4:00PM, 7:00PM]), we reset the time window to its initial interval, increase its size by one hour, and repeat. This effectively defines a neighbourhood selection heuristic that increases in size over time.

6   Experimental Results and Analysis

In this section we present a systematic experimental analysis of our initial models as well as the search manipulation results for our CP approach. All experiments are implemented in C++ on a hexacore machine with a Xeon processor and 16 GB of RAM running Mac OS X Yosemite. We use CP Optimizer (for CP) and CPLEX (for MIP) from the IBM ILOG CPLEX Optimization Studio version 12.6.2 single-threaded for all simulations with default search and inference settings unless otherwise noted.

Problem Instances. We consider five instances sizes defined based on the number of robots, $|R|$, and users, $|U|$. These sizes are: 2×5, 2×10, 3×15, 3×20, 4×25. These sizes are selected to reflect real-world retirement home problems. For each problem size we produce 10 unique instances, resulting in an instance set of 50 problems. Transition matrices, Δ, for each instance are based on randomly generated travel distances between locations within the facility, satisfying the triangle inequality $(\delta_{(a,b)} + \delta_{(b,c)} \geq \delta_{(a,c)})$. Two mandatory break periods (in addition to mealtimes) are randomly inserted into user calendars, each with a duration ranging from 30 min to one hour and a randomly assigned start time. We use $|G| = UB_1 = 5$ available bingo game tasks and $|C_k| = UB_2 = 3, \forall r_k \in R$, available recharge tasks. We use a 5 min run-time limit for all experiments, unless otherwise noted.

Initial CP and MIP Performance. We ran experiments on the problem scenarios using the initial CP and MIP formulations presented in Sect. 4 with default solver settings. CP is able to find feasible solutions for 9/50 problem instances, as seen in Table 1, while the MIP formulation is not able to find any feasible solutions. The numerous MTZ sequencing constraints have a poor linear relaxation strength, largely due to the inclusion of large integer values (i.e., H and β_{max}) for disjunctive reasoning. We believe the poor MIP performance is due to the extensive reliance on these constraints; future work will involve looking at using generalized subtour elimination [31] to improve algorithm performance. These experiments strongly suggest that CP is more suitable, of the two methods, for solving this problem. This finding is in agreement with the conclusions of previous research on a similar single-robot variant of the problem [5].

Bounds on Solution Quality. In order to measure the performance of our methods, we make efforts to determine provable and non-trivial upper bounds on solution quality. Valid bounds can be determined using the best dual bound from MIP, however, for these problems MIP is unable to produce non-trivial bounds within run-times of one hour, even when the problem is relaxed, multithreading is permitted, and the search emphasis is set to focus on dual bound strengthening.

Instead, we use a relaxation of our initial CP formulation, noting that the relaxation is only a true bound if the solution is proven optimal. The relaxed formulation is as follows:

$$\left\{ max \sum_{u_i \in U} \sum_{j \in G} \texttt{Pres}(x_{ij}) \; : \; Constraints \; (2)-(8), (13)-(15), (35)-(36) \right\} \quad (37)$$

This formulation relaxes the energy component of the problem in both the objective and constraints, while the remainder of the constraints are enforced. The set of tasks available to a robot becomes $\bar{T}_k^r := T_k^r \setminus C_k$. Solver inference is adjusted to the highest level (*extended*) in order to increase the amount of propagation performed at each search node, and symmetry breaking constraints are included. With these considerations made, the above model is only able to solve the first ten instances (instance sizes 2×5 and 2×10) to proven optimality within a run-time limit of one hour. The remainder of the instances yield unproven upper bound estimations.

Numerical Results. We present the performance of the various CP-based approaches we have investigated in Table 1. $CP_{default}$ is the original formulation with default solver settings, $CP_{default+SB}$ adds the symmetry breaking constraints, CP_{SP_1} represents the best performing single-stage variable instantiation strategy, and $CP_{SP_{1\rightarrow2}}$ the best performing double-stage instantiation strategy. To implement grouped variable ordering heuristics within IBM ILOG CP Optimizer, we use *search phases*. We conduct 250 experiments (50×5) and 1,000 experiments (50×20), respectively, for the single and double-stage orderings, in efforts to deduce the best variable instantiation strategy. $CP + LNS$ represents the performance of our CP-based time-window LNS approach.

We use mean relative error (MRE) to measure performance, calculated as follows:

$$MRE_{(\Omega, 0.1)} = \sum_{f \in F} \frac{c^*(f) - c(\Omega, 0.1)}{|F| \times c^*(f)} \times 100 \quad (38)$$

where the MRE for a particular method Ω at 0.1 s, for example, is calculated as above. The solution $c^*(f)$ represents the upper bound (or upper bound approximation) obtained by solving the relaxed CP formulation as presented in Formulation (37) for one hour. Problem instances $f \in F$ represent problems for which feasible solutions are found in the 5 min run-time limit.

Table 1. CP approach results: mean relative error (%) over time. '†' indicates approximate bound, $c^*(f)$, used for calculation. A value of '-' indicates the approach failed to find a feasible solution for all ten instances at that run-time. '# Inf.' represents the number of instances for which no feasible plan was found after 300 s of run-time.

$\|R\| \times \|U\|$	Approach	Run-time (s)						
		0.1	1	5	10	100	300	# Inf
2×5	$CP_{default}$	91.2	73.1	36.5	35.4	22.3	20.1	2
	$CP_{default+SB}$	97.2	91.5	65.6	55.9	29.9	27.0	2
	CP_{SP_1}	**74.6**	50.7	18.4	15.1	**0.1**	**0.1**	0
	$CP_{SP_{1 \to 2}}$	89.7	**48.0**	**17.1**	**15.0**	1.3	**0.1**	0
	$CP + LNS$	**74.6**	50.7	18.4	15.1	**0.1**	**0.1**	0
2×10	$CP_{default}$	-	-	-	-	93.5	91.5	9
	$CP_{default+SB}$	-	-	-	-	-	-	10
	CP_{SP_1}	-	97.8	51.8	45.8	**20.6**	12.8	0
	$CP_{SP_{1 \to 2}}$	-	**89.9**	54.0	49.0	22.6	19.8	0
	$CP + LNS$	-	97.8	**51.5**	**45.1**	22.0	**2.8**	0
$3 \times 15^{\dagger}$	$CP_{default}$	-	-	-	-	-	-	10
	$CP_{default+SB}$	-	-	-	-	-	-	10
	CP_{SP_1}	-	99.6	83.2	53.7	32.9	21.7	0
	$CP_{SP_{1 \to 2}}$	-	**99.5**	**65.1**	**44.8**	29.4	25.9	0
	$CP + LNS$	-	**99.5**	82.5	53.3	**29.0**	**13.7**	0
$3 \times 20^{\dagger}$	$CP_{default}$	-	-	-	-	-	-	10
	$CP_{default+SB}$	-	-	-	-	-	-	10
	CP_{SP_1}	-	-	97.3	87.4	48.9	41.5	0
	$CP_{SP_{1 \to 2}}$	-	-	**91.6**	**76.3**	**41.2**	**34.5**	0
	$CP + LNS$	-	-	97.3	87.2	43.4	35.2	0
$4 \times 25^{\dagger}$	$CP_{default}$	-	-	-	-	-	-	10
	$CP_{default+SB}$	-	-	-	-	-	-	10
	CP_{SP_1}	-	-	**99.0**	**91.4**	**45.1**	36.1	0
	$CP_{SP_{1 \to 2}}$	-	-	96.2	86.0	47.5	39.8	0
	$CP + LNS$	-	-	99.3	92.1	**45.1**	**35.6**	0

The default CP solver settings struggle to find any solutions for instances beyond 2×5 in size. Furthermore, it would seem that the symmetry breaking constraints reduce performance. The initial CP formulation without symmetry breaking is able to find feasible solutions to one 2×10 instance, whereas the symmetry breaking model could not find feasibility for this problem size.

The single stage search phase that performed the strongest involved instantiating the robot bingo game facilitation variables, y_{kj} $\forall r_k \in R$; $i \in G$, first as was previously postulated. Somewhat surprisingly, however, is the substantial

impact on solver performance compared to the default settings, improving MRE by $\geq 20\%$ for smaller instances and even more for larger problems. Such an improvement in MRE translates to a proportional increase in social and cognitive activity participation throughout the course of the day. The best double-phase instantiation involved fixing user bingo game participation x_{ij} variables first, and then robot bingo game facilitation (as in single stage) variables second. Referring to the table, double-stage instantiation offers benefit in a number of areas, particularly for the instances involving one and three robots in time limits ranging from 5 to 100 s. We note that both instantiation methods find solutions with just 0.1 % MRE for the first instance size. It appears that it becomes difficult to exploit problem structure past the assignment of bingo game tasks for this MRTA problem. Instantiating other groups of variables first (e.g., recharge tasks or reminders) resulted in performance similar to the CP default settings, although still somewhat stronger.

Due to the first minute spent finding an incumbent, CP with large neighbourhood search (LNS) has better performance in the later run-time limits. The performance of the method is notably strong at the 300 s run-time limit for instances 2×10 and 3×15, where it outperforms the search phases by a significant $\geq 12\%$ MRE. The LNS method uses an instantiation strategy very similar to CP_{SP_1} for the first minute, and as such the values are very similar for those run-times. LNS successfully finds the best solution for the largest problem by the run-time limit, outperforming both search phase methods.

7 Conclusions and Future Work

We applied constraint programming (CP) and mixed-integer programming (MIP) to the planning and scheduling of multiple social robots within a retirement home. This problem required task allocation and scheduling, aiming to maximize bingo game participation, while minimizing an energy consumption component. The proposed approaches reason about disjoint time windows, cross-schedule precedence relationships, spatial transition times of both users and robots within the environment, and robot energy consumption/replenishment.

Initial numerical experiments using default solver settings indicate that CP significantly outperforms MIP for the studied problem. We present methods for further enhancing CP performance through search manipulations, as well as a method for generating provable bounds for this problem by solving a relaxed CP formulation. Specifically, we investigated single and double-stage grouped variable ordering heuristics, concluding that instantiating the variables related to bingo game facilitation first has high positive impact on solution quality, significantly outperforming the default settings of the CP solver. We also implement a large neighbourhood search (LNS) using a time window variable selection heuristic. This method significantly outperforms the other approaches for mid-sized instances, and yields the strongest performance on the largest instances within the run-time limit. Due to these promising results, we plan to investigate alternative LNS procedures in future work.

Overall, results indicate that CP is a promising technology for our retirement home application. The next step is to move from simulation-based experimentation to deployment on real robots. We are integrating the CP solver into our robot architecture for field trials. In parallel, we are continuing research on the use of constraint programming in rescheduling and replanning as this will be a key functionality of the deployed system.

References

1. De Luca, A.E., Bonacci, S., Giraldi, G.: Aging populations: the health and quality of life of the elderly. La Clinica Terapeutica **162**(1), e13-8 (2010)
2. Francesca, C., Ana, L.-N., Jérôme, M., Frits, T.: OECD Help, Health Policy Studies Wanted? Providing, Paying for Long-Term Care: Providing and Paying for Long-Term Care, vol. 2011. OECD Publishing (2011)
3. Bemelmans, R., Gelderblom, G.J., Jonker, P., De Witte, L.: Socially sassistive robots in elderly care: a systematic review into effects and effectiveness. J. Am. Med. Directors Assoc. **13**(2), 114–120 (2012)
4. Louie, W.-Y.G., Vaquero, T., Nejat, G., Beck, J.C.: An autonomous assistive robot for planning, scheduling and facilitating multi-user activities. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 5292–5298. IEEE (2014)
5. Booth, K.E.C., Tran, T.T., Nejat, J.G., Beck, C.: Mixed-integer, constraint programming techniques for mobile robot task planning. Robot. Autom. Lett. **1**(1), 500–507 (2016)
6. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems, vol. 39. Springer Science & Business Media, US (2012)
7. Rossi, F., Van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
8. Hooker, J.N., Ottosson, G.: Logic-based benders decomposition. Math. Program. **96**(1), 33–60 (2003)
9. Achterberg, T., Berthold, T., Koch, T., Wolter, K.: Constraint integer programming: a new approach to integrate CP and MIP. In: Trick, M.A. (ed.) CPAIOR 2008. LNCS, vol. 5015, pp. 6–20. Springer, Heidelberg (2008)
10. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
11. Laborie, P., Godard, D.: Self-adapting large neighborhood search: application to single-mode scheduling problems. In: Proceedings MISTA 2007, Paris, pp. 276–284 (2007)
12. Parker, L.E.: L-alliance: task-oriented multi-robot learning in behavior-based systems. Adv. Robot. **11**(4), 305–322 (1996)
13. Botelho, S.C., Alami, R.: M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In: Proceedings of the 1999 IEEE International Conference on Robotics and Automation, vol. 2, pp. 1234–1239. IEEE (1999)

14. Dias, M.B., Stentz, A.: Traderbots: a market-based approach for resource, role, and task allocationin multirobot coordination. Robotics Institute, Carnegie Mellon University, Pittsburgh,PA, Tech. Rep. CMU-RI-TR-03-19 (2003)
15. Gerkey, B.P., Matari, M.J.: Sold!: auction methods for multirobot coordination. IEEE Trans. Robot. Autom. **18**(5), 758–768 (2002)
16. Liu, L., Michael, N., Shell, D.: Fully decentralized task swaps with optimized local searching. In: Proceedings of Robotics: Science and Systems (2014)
17. Korsah, G.A., Kannan, B., Browning, B., Stentz, A., Dias, M.B.: xbots: an approach to generating and executing optimal multi-robot plans with cross-schedule dependencies. In: 2012 IEEE International Conference on Robotics and Automation (ICRA), pp. 115–122. IEEE (2012)
18. Van Hentenryck, P., Saraswat, V.: Strategic directions in constraint programming. ACM Comput. Sur. (CSUR) **28**(4), 701–726 (1996)
19. Nareyek, A., Freuder, E.C., Fourer, R., Giunchiglia, E., Goldman, R.P., Kautz, H., Rintanen, J., Tate, A.: Constraints and AI planning. IEEE Intell. Syst. **20**(2), 62–72 (2005)
20. Goldman, R.P., Haigh, K.Z., Musliner, D.J., Pelican, M.J.S.: Macbeth: a multiagent constraint-based planner [autonomous agent tactical planner]. In: Proceedings of the 21st Digital Avionics Systems Conference, vol. 2, p. 7E3-1. IEEE (2002)
21. Doniec, A., Bouraqadi, N., Defoort, M., Le, V.T., Stinckwich, S.: Distributed constraint reasoning applied to multi-robot exploration. In: 21st International Conference on Tools with Artificial Intelligence, ICTAI 2009, pp. 159–166. IEEE (2009)
22. Broekens, J., Heerink, M., Rosendal, H.: Assistive social robots in elderly care: a review. Gerontechnology **8**(2), 94–103 (2009)
23. Vaquero, T., Mohamed, S.C., Nejat, G., Beck, J.C.: The implementation of a planning and scheduling architecture for multiple robots assisting multiple users in a retirement home setting. In: Artificial Intelligence Applied to Assistive Technologies and Smart Environments (AAAI 2015) (2015)
24. Laborie, P.: IBM ILOG CP optimizer for detailed scheduling illustrated on three problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 148–162. Springer, Heidelberg (2009)
25. Schneider, M., Stenger, A., Goeke, D.: The electric vehicle-routing problem with time windows and recharging stations. Transp. Sci. **48**(4), 500–520 (2014)
26. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. J. ACM (JACM) **7**(4), 326–329 (1960)
27. Drexl, M.: Synchronization in vehicle routing-a survey of VRPS with multiple synchronization constraints. Transp. Sci. **46**(3), 297–316 (2012)
28. Louie, W.-Y.G., Li, J., Vaquero, T., Nejat, G.: A focus group study on the design considerations, impressions of a socially assistive robot for long-term care. In: 2014 RO-MAN: The 23rd IEEE International Symposium on Robot, Human Interactive Communication, pp. 237–242. IEEE (2014)
29. Fahle, T., Schamberger, S., Sellmann, M.: Symmetry breaking. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 93–107. Springer, Heidelberg (2001)
30. Carchrae, T., Beck, J.C.: Principles for the design of large neighborhood search. J. Math. Mod. Algorithms **8**(3), 245–270 (2009)
31. Booth, K.E.C., Tran, T.T., Beck, J.C.: Logic-based decomposition methods for the travelling purchaser problem. In: Quimper, C.-G., Cavallo, M. (eds.) CPAIOR 2016. LNCS, vol. 9676, pp. 55–64. Springer, Heidelberg (2016). doi:10.1007/978-3-319-33954-2_5

Optimal Performance Tuning in Real-Time Systems Using Multi-objective Constrained Optimization

Stefano Di Alesio$^{(\boxtimes)}$

Certus Centre for Software Verification and Validation,
Simula Research Laboratory, Lysaker, Norway
`stefano@simula.no`

Abstract. Real-Time Embedded Systems (RTES) in safety-critical applications have to meet strict performance requirements to be deemed safe for operation. The satisfaction of these requirements at runtime often depends on configuration parameters that regulate how software tasks interact with hardware sensors and actuators. Tuning performance-related parameters is usually a manual, time-consuming, and error-prone process. This is because these parameters and their values define a large space of system configurations, and evaluating how each configuration affects the performance often requires executing the whole system. In this paper, we express RTES performance tuning as a multi-objective Constrained Optimization Problem (COP) over the configuration space that captures the dependencies between configuration parameters and performance requirements. In this way, the COP solutions characterize configurations predicted to maximize the satisfaction of performance requirements, and can in turn be used as guidelines for optimal performance tuning. We develop the COP as an OPL model for IBM ILOG CP OPTIMIZER, and validate our approach on a safety-critical I/O drivers system from the maritime and energy domain. The validation shows that our approach identifies within half an hour configurations characterized by tasks delay times that minimize deadline misses, response time, and CPU usage.

1 Introduction: Performance Tuning in Safety-Critical Systems

Failures in safety-critical systems, such as those in the energy, transport, and healthcare domains, could result in catastrophic consequences [18]. Therefore, the safety-related software components of these systems are usually subject to strict performance requirements involving real time and resources utilization constraints [27]. In particular, three performance requirements that are commonplace in safety-critical systems concern *task deadlines*, *response time*, and *CPU usage* [25]. Specifically, task deadlines state that the system tasks should always terminate before a given completion time, entailing that even a single deadline

miss severely compromises the system operational safety. Response time requirements specify that, in order for the outputs to be valid, the system should react to external inputs within a specified time. Finally, CPU usage constraints state that the system should always keep a certain percentage of free CPU time, to avoid that high computational load prevents the system from timely responding to safety-critical alarms.

However, safety-critical systems are progressively relying on Real-Time Embedded Systems (RTES), where software applications interact with the environment through sensors and actuators [23]. In complex RTES, the software components communicate with a large number of different devices. In particular, RTES have to ensure a smooth data transfer between hardware devices and software components. This is especially true in safety-critical systems, where external data should always be processed in brief time to guarantee a prompt reaction to critical events [35]. Therefore, the timing of RTES tasks can be configured to correctly operate with the specific devices connected. Nonetheless, tuning these timing properties without violating performance requirements is complicated by two main factors [17]. First, the task parameters related to temporal properties, such as delay times, offsets, and periods, range in a large domain of values. Second, the impact specific parameter values have over the system performance is hard to evaluate without executing the whole system. This is because RTES typically run on a preemptive Real-Time Operating System (RTOS) which may preempt a task execution in order to run an incoming higher priority task. Therefore, a minimal variation in a single task timing may trigger unpredictable interactions between other tasks [11].

As a consequence, it is often practice in industry to tune parameters related to RTES performance manually, based on the engineers expertise and knowledge of the system. This renders the process of tuning these performance-related parameters significantly time-consuming and error-prone [42]. Traditionally, systematic approaches for analyzing the RTES performance properties rely on Scheduling Theory [37], which is often based on unrealistic assumptions on the target system [3]. On the other hand, Model Checking [1] has been successfully proposed as an alternative for performance analysis and tuning [10], even though its scalability has to be further investigated due to the well-known state explosion problem [9]. Approaches based on metaheuristic search have also been proposed for identifying configuration parameters likely to satisfy performance requirements on CPU usage [29]. However, previous work in the field of software stress testing [14] suggests that complete search strategies, such as those based on Constraint Programming (CP), can potentially find solutions closer to the global optimum than meta-heuristics such as Genetic Algorithms (GA), and hence are worth being considered also in the context of performance tuning.

In this paper, we propose a methodology, based on Constrained Optimization, to help engineers tune performance-related parameters of RTES. The key idea behind our work is to identify scenarios where tasks finish their execution as far as possible from their deadlines, and exhibit low response time and CPU usage. Such scenarios are determined by the way tasks are scheduled to execute

at runtime by the RTOS. The task schedules depend in turn on the value of system timing parameters, on constraints derived from software design, and on the execution platform. Therefore, we propose a strategy to find combinations of timing properties that maximize the satisfaction of performance requirements on deadline misses, response time, and CPU usage. We characterize each of these combinations by a set of task *delay times*, and we refer to each set of delay times as a *configuration*.

2   Motivating Case Study: The Fire and Gas Monitoring System

The motivation behind our work originates from a case study in the maritime and energy domain concerning a Fire and Gas Monitoring System (FGMS). The system monitors potential gas leaks in off-shore oil extraction platforms, displaying to human operators data coming from smoke, heat, and gas flow sensors. In case a fire is detected, the FGMS triggers audio/visual alarms, activates the fire sprinklers, shuts down ongoing processes, and isolates electrical equipment. The software architecture of the system consists of drivers and control modules, as shown in Fig. 1a. Drivers support I/O communication between the software components and the operating environment, which consists of hardware sensors and actuators. Control modules implement the application logic of the FGMS, processing operational commands coming from the environment and accordingly deciding the actions to perform. The FGMS software components are executed by the RTOS VxWorks[1], which is configured with a fixed-priority preemptive scheduling policy on a tri-core computing platform.

We point out three main context factors that influence our formulation of the performance tuning problem in the FGMS. (1) In this paper, we do not consider FGMS-level performance requirements, which require considering interactions between drivers, control modules, and external hardware. On the other hand, we limit our scope to driver-level requirements, for which it is only necessary to consider the drivers subsystem. To avoid confusion, in the rest of this paper we will refer to the FGMS drivers as the *system* under investigation. (2) Different instances of a given driver are independent, i.e., they do not communicate with one another and do not share memory. For this reason, in this paper we focus on a single driver instance and do not consider interactions between them. (3) The FGMS performance profiling logs indicate that task deadlines, response time, and CPU usage of the drivers are not significantly affected by memory allocation activities such as garbage collection and data transfer operations on storage peripherals. For this reason, we do not consider the impact of memory usage on the drivers performance.

Drivers in the FGMS share the same design pattern, consisting of two periodic tasks, (*PullData* and *PushData*), and one singular task (*IODispatch*), which is executed only once during the drivers execution. The tasks communicate

---

[1] http://www.windriver.com/products/vxworks.

(a) Software architecture of the FMS   (b) Typical drivers data transfer scenario
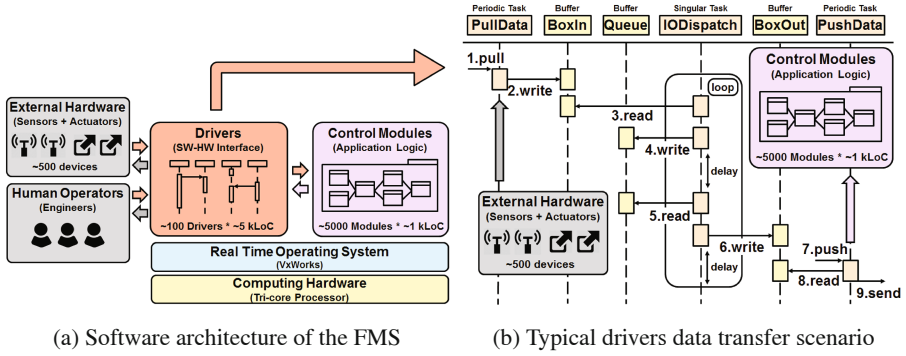
Fig. 1. Description of the Fire and Gas Monitoring System

through three fixed capacity buffers with mutually exclusive access, namely *BoxIn*, *Queue*, and *BoxOut*. Figure 1b shows how the three tasks collaborate in the typical operating scenario, that is a unidirectional data transfer between hardware sensors and control modules. (1) *PullData* periodically receives data from sensors or human operators through the *pull* signal, formats the data in an appropriate command form, and (2) writes it in *BoxIn*. (3) *IODispatch* reads the buffer, extracts the commands from the data, and (4) stores them in the priority *Queue*. After a given delay time, (5) *IODispatch* reads the highest priority command and (6) writes it to *BoxOut*. When the periodic *push* signal (7) activates *PushData*, the task (8) reads the commands from *BoxOut* and finally (9) sends them to the control modules for processing. Note that *IODispatch* is executed when the drivers are initialized, and encloses within an infinite loop four sequential read/write *activities*. In particular, the two activities writing in *Queue* and *BoxOut* are followed by delay times, implemented as *sleep* calls in the drivers source code. In particular, the delay times may vary during the *IODispatch* loop iterations, allowing the drivers to send data to the control modules at a variable rate. This design is meant to ensure that the FGMS data flow is slow enough to avoid overloading the computational resources, but fast enough to ensure prompt reaction to critical events.

The data transfer functionality is subject to strict performance requirements. Specifically, in each driver instance, (1) no task should miss its deadline, (2) the response time should be less than one second, and (3) the average CPU usage should be below 20 %. The main variables determining whether or not these requirements will be satisfied at runtime are the delay times after the write activities of *IODispatch*. Indeed, if the delay times are too short, *IODispatch* is running continuously and keeps the CPU busy, eventually exceeding the 20 % usage threshold. On the other hand, if the delay times are too large, *PullData* may fill up *BoxIn*, and be blocked waiting for *IODispatch* to empty the buffer. As a result, *PullData* is not able to terminate before the next *pull* signal arrives, missing its deadline. In general, it is hard to predict whether a set of delays in *IODispatch* will break deadlines in other tasks, or will make the driver exhibit

high response time or CPU usage at runtime. This is because the delay determines the arrival time of the activities in *IODispatch*, which in turn can preempt or be preempted by other tasks. Note that, however, the delay times of the *IODispatch* iterations are tunable parameters that engineers can set when configuring the drivers.

3 Related Work

The increasing complexity in RTES software and hardware architecture renders analyzing and estimating performance properties in RTES increasingly challenging [27], especially in safety-critical domains, where performance issues can impact the system behavior more than incorrect functionality [40]. In RTES, performance properties have been traditionally analyzed through verification approaches, such as Scheduling Theory [37] and Model Checking [1]. Theorems from Scheduling Theory are limited to providing sufficient or necessary conditions for a set of tasks to be schedulable, and are often based on unrealistic assumptions on the target system [3]. On the other hand, Model Checking approaches analyze time-related properties, such as task deadlines and resource usage, by proving reachability properties in state machines [10]. However, Model Checking requires complex formal modeling of the system, which is not always available for large systems and often leads to the state explosion problem [9]. Our experience suggests that, in several industrial contexts, RTES are developed by relatively small-sized teams consisting of developers with several years of expertise in their domain. This development strategy increases productivity by reducing the communication overhead [16], but can potentially come at the cost of overlooking the need of systematic performance analysis. This usually happens in systems with long lifespan, whose core functionalities undergo only minor updates over the years and are hence deemed stable. In such systems, performance tuning is mostly addressed by human expertise, which in turn relies on profiling and benchmarking tools that dynamically analyze performance properties [22]. Such tools, however, can only provide a rough performance assessment limited to a small number of system executions, which have to be manually investigated [42].

The use of search-based approaches to find optimal configuration parameters originates from the domain of control systems [39], where Genetic Algorithms (GA) have been applied to tune the performance of Proportional-Integrative-Derivative (PID) controllers [20]. In the context of Real-Time Systems, GA have been used to generate scenarios characterized by reproducible environmental conditions that push the system to break task deadlines [6]. These approaches have inspired the use of metaheuristic search to derive configuration parameters that are predicted to minimize the CPU usage [29]. In particular, Non-dominated Sorting GA (NSGA) have been used to identify task offsets that yield an optimal trade-off between CPU usage and requirements specifying groups of tasks that have to be executed within short time [30]. Even though these approaches have only been applied in the context of Static Cyclic Scheduling (SCS), which is non-preemptive, they represent the closest related work to that presented in this paper.

Previous work [14] in the area of stress testing suggests that, when compared to GA, Constraint Programming (CP) can find task schedules closer to the global optimum, and is hence worth investigating also in the context of performance tuning. For schedulability analysis, CP approaches [4] have been used since long time, especially in the domain of job-shop scheduling [26]. Among those, several approaches target task real-time constraints such as task deadlines [19], or timeliness [28]. Preemptive scheduling problems have also been approached with pure CP [7], and with hybrid approaches combining CP with GA [43]. The most recent implementations have successfully used both CP and Mixed Integer Programming (MIP) to solve priority-driven scheduling problems, albeit not addressing task preemption [24]. However, we are unaware of CP approaches targeted to the generation of software configurations predicted to satisfy performance requirements, and in particular of approaches addressing all the complexities of RTES such as multi-core architectures, task dependencies and triggering relationships, and priority-based preemptive scheduling policies.

4   Performance Tuning with Constrained Optimization

The approach presented in this paper extends earlier work [13,31] for deriving test cases exercising CPU usage and task deadlines requirements of multi-core RTES. Specifically, the approach has been adapted to derive configurations characterized by task delay times that maximize the satisfaction of requirements on their deadlines, response time, and CPU usage. In particular, we cast the search for these delay times as a multi-objective Constraint Optimization Problem (COP). The COP models a preemptive priority-based task scheduler with fixed priorities, task triggering, and dependencies on shared resources. The COP is derived from earlier work on generating stress test cases for RTES [15], and the key idea behind its formulation relies on four main points. (1) We model the system design, which is static and known prior to the analysis, as a set of constants. The system design consists of the tasks of the real-time application, their dependencies, offsets, periods, durations, deadlines, and priorities. (2) We model the system properties that depend on runtime behavior, and those that are configurable parameters, as a set of variables. The main real-time property in the first category is the specific runtime schedule of the tasks. Configurable properties, which are the output variables of the model, are instead the delay times between task activities. (3) We model the RTOS scheduler as a set of constraints among such constants and variables. Indeed, the RTOS scheduler periodically checks for triggering signals of tasks and determines whether tasks are ready to be executed or need to be preempted. (4) We model the performance requirements the system must satisfy (i.e., task deadlines, response time, or CPU usage) as objective functions to be minimized. By design, each solution of our COP is a sequence of task delay times, which in turn characterizes a configuration.

Our analysis is subject to two assumptions. (1) The RTOS scheduler checks the running tasks for potential preemptions at regular and fixed intervals of time,

called *time quanta*. Therefore, each time value in our problem is expressed as a multiple of a time quantum. (2) The interval of time in which the scheduler switches context between tasks is negligible compared to a time quantum. We found these two assumptions to be commonplace in several RTES, as the scheduling rate of operating systems varies in the range of few milliseconds, while the time needed for context switching is usually in the order of nanoseconds [33]. These assumptions allow us to consider time as discrete in our analysis, and model the COP as an Integer Program (IP) over finite domains.

We implemented the COP in the Optimization Programming Language (OPL) [38], and solved it with IBM ILOG CPLEX CP OPTIMIZER[2]. This choice was motivated by practical reasons, such as its extensive documentation, strong supporting community, and acknowledged efficiency to solve optimization problems. Note that we could not express a preemptive priority-driven scheduling problem in an effective way that exploited the solver capabilities of working with task intervals [8], and hence we implemented our COP as a traditional IP. In the following, we describe our constraint model (Sect. 4.1), and how to use it to model infinite loops of activities separated by a delay time (Sect. 4.2).

4.1 Description of the Constrained Optimization Problem

Constants. As explained before, we consider time as discretized in time quanta. Therefore, we define the *observation interval* T as an integer interval of length tq, i.e., $T \stackrel{\text{def}}{=} [0, tq - 1]$, representing the time interval during which we observe the system behavior. Each time value $t \in T$ is a time quantum. We define c as the *number of cores* in the execution platform, representing the maximum number of tasks that can be executed in parallel, J as the *set of tasks* of the system, and R as the *set of resources* shared by such tasks. Each resource $r \in R$ is typically implemented as a buffer, and serves as a mechanism to store data for synchronous and asynchronous communication between tasks. Each task $j \in J$ has a set of static properties, whose values are part of system design and known prior to the execution of the system. These static properties are defined in Real-Time Scheduling Theory [34], and comprehend the task *priority* pr_j, *period* pe_j, *offset* of_j, *deadline* dl_j, and *number of task executions* te_j. In particular, we refer to the kth execution of task j as the couple $(j, k-1)$. In this way, the first execution of j is the couple $(j, 0)$. The offset and period determine the number of task executions so that $te_j = \left\lfloor \frac{tq - of_j}{pe_j} \right\rfloor$. For simplicity, we define the interval K_j of executions of task j as $K_j \stackrel{\text{def}}{=} [0, te_j - 1]$, so that, in the context of a given j, $k \in K_j$. We also consider the *duration* dr_j of tasks as a constant equal to the task Worst Case Execution Time (WCET), which can be estimated through different techniques both statically, using the system design, and dynamically, by measuring execution times [41]. In our context, the WCET is estimated by selecting the worst-case time across several executions of the system. Note that considering the duration of each task as its WCET is a common practice when

---

[2] http://www.ibm.com/software/commerce/optimization/cplex-cp-optimizer/.

analyzing task real-time properties [17]. We refer to the dth time quantum of the task execution (j, k) as the triple $(j, k, d-1)$. For simplicity, we also define the interval D_j of duration time quanta of a task as $D_j \stackrel{\text{def}}{=} [0, dr_j - 1]$, so that, in the context of a given j, $d \in D_j$. Finally, we also define as constants the tasks *triggering relation tg*, and *read (write) dependency relation rd (wr)*. The former is an irreflexive and antisymmetric binary relation over $J \times J$, where tg_{j_1, j_2} holds if the event triggering j_2 occurs when j_1 finishes its execution, plus a possible delay. The latter are binary relations over $J \times R$, where $rd_{j,r}$ $(wr_{j,r})$ holds if j reads data from (writes data to) r during its execution. Note that tasks in a dependency relation cannot be executed in parallel nor can preempt each other, but one can execute only after the other has released the lock on the resource.

Variables. Tasks in J also have a set of dynamic properties, whose values depend on the runtime behavior of the system, and hence are not known prior to the analysis. Indeed, the values for these variables are calculated during the search, and represent the output data of the COP. In the context of constraint solving, variables whose domain values define the search space are said to be *independent* or *decision variables* [21]. Indeed, the goal of a constraint solver is to assign values for the independent variables that satisfy all the constraints, optimizing an objective function when specified. In our model, the independent variables characterize configurations in terms of the delay time that trigger the task executions. The independent variables of our model, marked with a single dot $(\cdot)$, are the time quanta $\dot{ac}_{j,k,d}$ where the system tasks are active and executing, the arrival times $\dot{at}_{j,k}$ of triggered task executions, and their delay times $\dot{dy}_{j,k}$. All these variables have domain in T. In particular, we refer to the set of all $\dot{ac}$ variables as the *schedule* produced by the arrival times of tasks in J. Note that the arrival times of periodic tasks are constant, and determined by the task period and offset: $\dot{at}_{j,k} = of_j + k \cdot pe_j$. In addition to these independent variables, we also define *dependent* variables, whose value is defined by a mathematical expression of independent variables and constant values. Dependent variables, marked with a double dot $(\cdot\cdot)$, simplify our notation by allowing us to easily formulate constraints and objective functions. For example, we define as dependent variable the *start* and *end time* of tasks $\ddot{st}_{j,k}$ and $\ddot{en}_{j,k}$, i.e., the first and the last time quantum in which (j, k) is executing: $\ddot{st}_{j,k} \stackrel{\text{def}}{=} \dot{ac}_{j,k,0}$ and $\ddot{en}_{j,k} \stackrel{\text{def}}{=} \dot{ac}_{j,k,dr_j-1} + 1$. In particular, the end times of tasks allows to define the *deadline miss* $\ddot{dm}_{j,k}$ of a task execution as the amount of time by which (j, k) missed its deadline: $\ddot{dm}_{j,k} \stackrel{\text{def}}{=} \ddot{en}_{j,k} - (\dot{at}_{j,k} + dl_j)$. We also define the *system load* $\ddot{ld}_t$ as the number of tasks active at time t: $\ddot{ld}_t \stackrel{\text{def}}{=} \sum_{j,k,d} (\dot{ac}_{j,k,d} = t)$. Note that $(\ddot{ac}_{j,k,d} = t)$ is a boolean variable that is evaluated to 1 when true, and to 0 when false. Furthermore, the dependent variables include the *preempted time quanta* $\ddot{pm}_{j,k,d}$ of task executions, defined as the number of time quanta for which (j, k) is preempted for the dth time: $\ddot{pm}_{j,k,d} \stackrel{\text{def}}{=} \dot{ac}_{j,k,d} - \dot{ac}_{j,k,d-1} - 1$, and the *waiting time* $\ddot{wt}_{j,k}$ of task executions, defined as the amount of time for which (j, k) has to wait after its arrival time before starting its execution: $\ddot{wt}_{j,k} \stackrel{\text{def}}{=} \ddot{st}_{j,k} - \dot{at}_{j,k}$. The preempted time quanta and the waiting time allow us

to easily formulate constraints specifying that tasks should only be preempted by higher priority tasks, and should postpone their starting time only when they are locked on a shared resource, or waiting for data to be written, or because there is no processing core available. Finally, we define the *resource status* indicator $\ddot{rs}_{r,t}$ as a binary variable indicating whether the resource r is full or empty at time t. For any pair of executions of two tasks j_1 and j_2 which respectively read and write r, $\ddot{rs}_{r,t}$ has value 1, i.e., the resource is full and ready to be accessed for read operations, between the end of j_2 and the end of j_1, and has value 0, i.e., the resource is empty and cannot be accessed for read operations, otherwise: $\forall j_1, j_2 \in J, k \in K_{j_1} \cap K_{j_2}, r \in R, t \in T \cdot rd_{j_1,r} \wedge wr_{j_2,r}$

$$\ddot{rs}_{r,t} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } t \in [\ddot{en}_{j_2,k}, \ddot{en}_{j_1,k}] \\ 0 & \text{otherwise} \end{cases}$$

The resource status indicator allows us to easily formulate constraints specifying that tasks can only write to empty buffers, and read from full buffers.

Constraints. We define five sets of constraints which model task runtime interactions, i.e., locks and preemptions, and the way in which the RTOS scheduler executes these tasks based on their triggering and dependency relations. In this paper, we only report shortened expressions of constraints, labeled with the letter γ, as their rigorous mathematical formulation is part of previous work [15]. *Well-formedness constraints* specify relations among variables that directly follow from their definition in the schedulability theory. For example, well-formedness constraints state that each task execution starts after its arrival time, and ends after the task duration dr ($\gamma_1 : \dot{at}_{j,k} \leq \ddot{st}_{j,k} \leq \ddot{en}_{j,k} - dr_j$). Furthermore, note that resources may be shared between more than two tasks. This entails that more than one task execution can be locked on a given resource at any time. In RTES, task queues regulate the access of a resource by multiple locked tasks. In our COP, these task queues are modeled through a well-formedness constraint stating that if task j_2 is ready to be executed at the same time as a lower-priority task j_1, j_2 starts first: ($\gamma_2 : \dot{at}_{j_1,k_1} = \dot{at}_{j_2,k_2} \longleftrightarrow \ddot{st}_{j_2,k_2} \leq \ddot{st}_{j_1,k_1}$). *Temporal ordering constraints* specify the relative ordering of tasks based on their dependency and triggering relations. In particular, these constraints state that the a task j_2 triggered by j_1 arrives after the delay of j_1, counted from when j_1 ends ($\gamma_3 : \ddot{en}_{j_1,k} + \dot{dy}_{j_1,k} = \dot{at}_{j_2,k}$). Furthermore, temporal ordering constraints state that the executions of two dependent tasks j_1 and j_2 cannot overlap, i.e., that one can only start after the other has ended ($\gamma_4 : \ddot{en}_{j_1,k_1} \leq \ddot{st}_{j_2,k_2} \vee \ddot{en}_{j_2,k_2} \leq \ddot{st}_{j_1,k_1}$). Finally, these constraints state that (1) a task cannot write to a full buffer, i.e., that the start time of a task j writing on a resource r has to occur when r is empty ($\gamma_5 : \ddot{rs}_{r,\ddot{st}_{j,k}} = 0$), and that (2) a task cannot read from an empty buffer, i.e., that the start time of a task j reading from a resource r has to occur when r is full ($\gamma_6 : \ddot{rs}_{r,\ddot{st}_{j,k}} = 1$). *Multi-core constraints* capture the concurrent nature of the computing platform, stating that no more than c tasks can be active at any time ($\gamma_7 : \ddot{ld}_t \leq c$). *Preemption constraints* capture the priority-driven preemptive scheduling of the RTOS, stating that each task should be preempted

when a higher priority task is ready to be executed and no cores are available. Finally, *scheduling efficiency constraints* ensure that tasks are not preempted unnecessarily and are executed as soon as possible.

Objective Functions. We formalize three objective functions representing task deadlines, response time, and CPU usage. The functions are minimized in a multi-objective optimization problem, in a way that solutions of the COP characterize scenarios approaching optimal tradeoffs between the objective values. Note that, even though the performance requirements specify a maximum threshold on the value of task deadlines, response time, and CPU usage, the value of these properties is not bound by any constraint in the COP. Therefore, the search process might initially find solutions that satisfy the constraints, but whose objective value is greater than the threshold expressed by the requirements. However, our COP is based on estimates of the tasks WCET, which might be over-pessimistic. For this reason, the delay times characterized by these solutions might not violate the performance requirements at runtime, and hence are worth looking at during configuration. Nevertheless, the COP minimizes the objective functions representing the performance requirements, because the lower the objective values, the higher the confidence that the system achieves a satisfactory performance. We define the *CPU usage function* F_{CU} that models the system CPU usage: $F_{CU} \stackrel{\text{def}}{=} \sum_{t \in T} (\ddot{ld}_t > 0)/tq$. F_{CU} measures the average CPU usage of the system over T as the percentage of T where at least one core is busy. We define the *deadline misses function* F_{DM} that models the requirement on task deadlines: $F_{DM} \stackrel{\text{def}}{=} \sum_{j,k} 2^{\ddot{dm}_{j,k}}$. To ensure that tasks completing in short time do not overshadow deadline misses, F_{DM} assigns to $\ddot{dm}$ an exponential contribution towards the sum [14]. Recall that $\ddot{dm}_{j,k}$ is positive if the task execution (j, k) misses its deadline, and negative otherwise. Finally, we also define the *response time function* F_{RT} that models the system response time. In traditional scheduling, the response time measures the maximum length in time quanta of the task schedule restricted to a single execution. This means that the response time is the maximum time between the kth arrival time of a task, and the kth end time of a possibly different task. The response time is also traditionally known as *makespan* [32].

$$F_{RT} \stackrel{\text{def}}{=} \max_{j_1, j_2 \in J,\ k \in K_{j_2} \cup K_{j_2}} \left(\ddot{en}(j_1, k) - \dot{at}(j_2, k) \right)$$

4.2   Modeling Task Activities and Infinite Loops

In task scheduling, a task j consists of a vector $[a_0, a_2, \ldots a_{n-1}]$ of n activities a_i executed sequentially. At the lowest level of abstraction, an activity is a single statement in a task source code. For this reason, several task properties defined in Sect. 4.1 can also be considered at activity-level. For example, the duration of an activity is its WCET, while its priority is equal to the priority of its task. In particular, the delay time of an activity a_i is the minimum time that has to

elapse, not considering preemptions, between the completion of a_i and the start of a_{i+1}. Since activities are executed sequentially, the arrival time of an activity a_i is the time when the preceding activity a_{i-1} finishes executing, plus the delay of a_{i-1}. Task interactions can also be considered at activity-level, as activities may depend on, or trigger other activities in different tasks. For instance, an activity may trigger another activity of a waiting task by sending a specific message to that task, or can launch a new task by triggering its first activity. Therefore, a task $j = [a_0, a_1, \ldots a_{n-1}]$ with priority p consisting of n activities a_i can be considered for scheduling purposes as a vector $[j_0, j_1, \ldots j_{n-1}]$ of n tasks with priority p, where the duration of j_i is equal to the duration of a_i, and where j_i triggers j_{i+1}. In this case, each task j_i inherits the dependencies and triggering relationships of the corresponding activity a_i. Note that this property holds under the assumption that the RTOS overhead for managing tasks in negligible compared to their execution and interarrival times. This assumption has proven to be realistic in most RTES [36].

Given this mapping between activities and tasks, we can model tasks enclosing activities in infinite loops, such as *IODispatch* (Fig. 1b). Consider the task $j = [a_0, \ldots a_{n-1}]$, where the n activities are enclosed in an infinite loop. j can be modeled through a vector of $n + 1$ tasks $[j'_0, j_0 \ldots j_{n-1}]$. In the vector, j'_0 and j_0 both correspond to a_0, and each other task j_i corresponds to the activity a_i. Each task in the vector has the same duration, priority, and dependencies of its corresponding activity. Each task triggers the following one forming a *triggering*

Fig. 2. Emulation of the loop in *IODispatch* through five tasks

chain, with the exception of j'_0 that triggers j_1, and j_{n-1} that triggers j_0. Note that, if all the activities in j are enclosed in an infinite loop, the task j'_0 is necessary in order to ensure that the COP is feasible. Consider indeed the alternative of modeling j through the tasks $j_0 \ldots j_{n-1}$, with each task triggering the following one and j_{n-1} triggering j_0. Recall that a triggered task execution arrives after when its triggering execution finishes, plus a possible delay. This is specified by the temporal ordering constraint γ_3 introduced in Sect. 4.1. Therefore, a circular dependency of tasks triggering each other would render the model infeasible, because the first arrival time of j_0 would depend from a previous execution of j_{n-1} that never happened. This means that the temporal ordering constraint above would result in a *non well-defined* recursion, i.e., a recursion with no base case. To overcome this issue, we model the first execution of a_0 as a separate task, namely j'_0. j'_0 is a *singular* task, i.e., a periodic task whose period is equal to the observation interval T, and hence is only executed once during the system execution. After finishing, j'_0 triggers j_1, emulating a_0 triggering for the first time a_1 in j.

Fixing the first arrival time of the first activity executed during the loop allows the solver to find the arrival times of subsequent activities by unrolling the task executions in the triggering chain. Figure 2 shows how the loop in *IODispatch* is modeled through five tasks, namely *Init*, *IOBoxRead* (*IOBR*), *IOQueueWrite* (*IOQW*), *IOQueueRead* (*IOQR*), and *IOBoxWrite* (*IOBW*). In the figure, the numbers within the rectangles on the lifeline show the correspondence between activities and tasks.

5   Industrial Experience: Context, Process, Results, and Discussion

The work reported in this paper originates from the collaboration over the years with Kongsberg Maritime (KM)[3], a leading company in the production of systems for positioning, surveying, navigation, and automation of merchant vessels and offshore installations. When developing the software components of their real-time systems, KM faces significant challenges which have motivated our research. Therefore, the main goal of our industrial evaluation is to investigate whether CP can effectively support performance tuning in an industrial context. This aspect depends on whether we can conveniently use the output of our analysis, i.e., the values for the delay time variables in the COP, to derive configurations that satisfy the system performance requirements. In particular, we investigate this practical usefulness through two main factors. First, we note how, for practical use, performance tuning has to accommodate time and budget constraints. Therefore, we analyze the *efficiency* of our approach, i.e., the time needed to generate delay times predicted to satisfy the performance requirements. Second, recall from Sect. 2 that requirements on task deadlines and response times often conflict with thresholds on the CPU usage, because it is hard to achieve shorter task completion times without over-utilizing the CPU. In practice, the goal of performance tuning in the FGMS is finding safe margins in which delay times yield a trade-off between conflicting performance requirements without violating them. For this reason, we also analyze the *effectiveness* of our approach, i.e., the capability of the generated delay times to lead to scenarios achieving such satisfactory trade-off between performance requirements.

Experimental Design. Recall from Sect. 2 that we characterize system configurations by delay times between activities in the *IODispatch* task of the FGMS drivers. Therefore, such delay times are the main independent variables in our constraint model (Sect. 4). We performed an experiment with the FGMS drivers using an observation interval T of five seconds, assuming, in accordance with the specification of the RTOS executing the FGMS, time quanta of 10 ms. The search for optimal solutions was driven by a lexicographic multi-objective optimization. In lexicographic ordering, the first criterion is considered as the most important one, so that its improvement is worth any loss on the other criteria. The second criterion is the second most important, so that only losses on the first

---

criterion are not allowed for its improvement, and so on. Using multi-objective optimization allows us to identify a Pareto-optimal frontier of solutions that are *non-dominated*, i.e., solutions x^* for which no other solution x exists such that x has a better objective value than x^* for *all* the criteria. The solutions in the frontier achieve an optimal trade-off between the search criteria, because any solution with a better objective value for one criterion has a worse objective value for at least another criterion. Investigating solutions in the Pareto frontier is particularly useful to evaluate trade-offs of conflicting optimization criteria, such as F_{RT} and F_{CU}.

We run our model for six times, one for each lexicographic permutation of F_{DM}, F_{RT}, and F_{CU}. Each run was performed on an Amazon EC2 *m2.xlarge* instance[4] with a timeout of two hours, after which the solver was instructed to terminate. We also recorded the computation times of the first solutions predicted to satisfy F_{DM}, F_{RT}, and F_{CU}. Consistent with the terminology used in Integer Programming, we refer to these (sub)optimal solutions as *incumbents* [2]. The COP consisted of approximately 500 variables and one million constraints, and used up to 10 GB RAM during resolution.

Results and Discussion — Efficiency. Figure 3 shows 18 graphs reporting the experimental results for the six runs. The graphs are organized in a matrix, where each row corresponds to an objective function (F_{CU}, F_{DM}, and F_{RT}, respectively), and each column corresponds to a run. Runs are reported in the format *XX-YY-ZZ*, where each group of two letters corresponds to an optimization criterion, with *XX* being the most important, *YY* being the second most important, and *ZZ* the least. In each graph, the x-axis reports the incumbent computation times in the format *hh:mm:ss*, and the y-axis reports the corresponding objective value. The graphs related to F_{CU} and F_{RT} also report an horizontal line representing the maximum threshold on the performance requirement. Note that, being defined as an exponential function of task deadline misses, F_{DM} has no threshold on its value. In each graph, we also highlight in a circle ($\bigcirc$) the first incumbent predicted to satisfy the relative performance requirement, and in a square ($\square$) the first incumbent predicted to satisfy *all* the requirements. For these incumbents, we report in a box their computation times and objective values. Finally, we report at the top right of each column the total number of solutions found in the run, and at the top center of each graph the number of incumbents satisfying the requirement. Recall from Sect. 4 that each solution of our COP is a sequence of task delay times.

To support engineers in configuring performance-related parameters of RTES, our approach should be able to efficiently produce usable results. In particular, engineers need to know for how long on average they should run our COP. The six runs found a total of 71 incumbents, terminating with proof of optimality in less than one hour when choosing F_{DM} as the primary optimization criterion (third and fourth column in Fig. 3). With the exception of F_{CU} in the runs *CU-DM-RT*, *CU-RT-DM*, and *DM-RT-CU*, the first solution predicted to satisfy any of the performance requirements was found in less than a minute.

In these three cases, the first solution predicted to exhibit a CPU usage less than 20 % was found approximately after 28, 27, and 15 min, respectively. In each run, the incumbents found presented no deadline misses. Overall, our COP was able to find solutions predicted to satisfy *at least* one performance requirement in less than one minute, and *all* the requirements in less than half an hour. In particular, the runs *DM-CU-RT*, *RT-CU-DM* and *RD-DM-CU* found the first solutions satisfying all the requirements in approximately 30 seconds, while the other runs did so in approximately 28 min. The delay times characterizing these solutions can be used to derive and test initial system configurations while the search continues, because the lower the objective value, the more likely the solutions are to satisfy the systems performance requirements. In summary, it is sufficient to run our COP for half an hour on the FGMS I/O drivers in order to find solutions satisfying all the requirements.

Results and Discussion — Effectiveness. As explained above, engineers are particularly interested in finding ranges of delay times where conflicting performance requirements are close to their thresholds, but are not violated. To find these ranges, we first have to identify the conflicting requirements by analyzing the trend of the objective functions. We note how, in each run, the objective value over time related to the first optimization criterion presents a monotonic decreasing trend. This is expected because each run performs a lexicographic optimization, for which any gain on the primary criterion is worth loss on the others. When looking into the trend of specific criteria over runs, F_{DM} shows no significant correlation with F_{CU} and F_{RT}. This seems counterintuitive, because tasks are likely to miss their deadlines if the response time is too long, and when the response time is short, tasks are likely to complete long before their deadline. However, even though the exponential shape of F_{DM} is very sensible to variations in task deadlines, the fluctuations in the objective value are several orders of magnitude smaller than the size of the observation interval T. We also note how F_{CU} and F_{RT} present an inversely proportional trend. Indeed, in the runs where F_{CU} is the first optimization criterion (first two columns in Fig. 3), F_{RT} tends to decrease over time, and vice versa (last two columns in Fig. 3). This is also expected because, as explained in Sect. 2, short delay times make *IODispatch* keep the CPU busy, while long delay times are likely to block *Pull-Data*, increasing the drivers response time. Therefore, when configuring delay times, it is necessary to analyze the trade-off between expected response time and CPU usage. Note that every incumbent found satisfies the requirement on task deadlines, and hence this trade-off analysis can ignore F_{DM}.

Figure 4 shows the Pareto-optimal frontier of F_{CU} and F_{RT}, whose solutions are highlighted with a solid bullet (●). The circle (○) highlights the first solution found in the frontier that satisfies all the requirements, for which we report computation time and objective values. Similar to Fig. 3, the two orthogonal lines represent the maximum threshold on F_{CU} and F_{RT}. Over the six runs, the search found ten solutions in the frontier, four of which satisfy the performance requirements. The first of such solutions was found in approximately 27 min, and corresponds to that highlighted in the *CU-RT-DM* run in Fig. 3. By definition,

Fig. 3. Objective values of F_{CU}, F_{DM}, and F_{RT} over time, grouped by run

the solutions in the frontier do not Pareto- dominate each other, entailing that for each solution in the frontier there does not exist any other solution with a lower CPU usage *and* a lower response time.

Therefore, the solutions in the frontier achieve an optimal trade-off between CPU usage and response time, and can be used to derive configurations that are as likely as possible to exhibit low CPU usage and response time. Finally, recall that the six solutions in the frontier above the 20 % CPU threshold might not violate such requirement at runtime due to pessimistic WCET estimates. These solutions are still worth being investigated, albeit with lower priority than the others. In particular, our experience suggests that the most valuable solutions lie in the extreme regions of the Pareto frontier, close to the highest value for a single objective function, and in the central part, where the performance require-

Fig. 4. Pareto-optimal frontier (solid bullets) of F_{CU} and F_{RT}

ments are equally far from their maximum values. In fact, solutions in extreme regions can be used to push the system performance to the limit, while solutions in the central area guarantee a balance between conflicting requirements.

6   Concluding Remarks

In this paper, we presented a multi-objective Constrained Optimization Problem (COP) for generating RTES configurations characterized by values of configurable timing properties that satisfy a set of performance requirements. In particular, we presented a COP whose solutions are task delay times that characterize scenarios where tasks are as far as possible from their deadlines, and exhibit low response time and CPU usage. However, we note that casting the scheduling analysis of RTES as a COP is a flexible strategy that can be tailored to support activities in different phases of software development, such as stress testing and performance tuning, as well as to suit different application scenarios. For example, in order to generate task offsets that satisfy a requirement on minimal throughput, we would only need to modify the existing COP by (1) specifying task offsets as variables, rather than constants, and (2) defining a new throughput objective function. As another example, to target a system with a priority ceiling scheduling policy, we would have to modify only the preemption constraints by specifying that tasks locking a resource shared with a high priority task cannot be preempted. These adaptations would be similar to that done in this paper with respect to previous work in the area of stress testing [15].

We validated our approach on a RTES from the maritime and energy domain concerning safety-critical device drivers, showing that our approach is able to find

Pareto-optimal solutions with respect to CPU usage and response time in less than half an hour. Recall that our approach builds also upon previous work in the context of performance analysis, which introduces a conceptual model to capture the timing and concurrency abstractions required to analyze response time and CPU Usage in RTES [31]. Those abstractions form the basis of both the COP presented in that work, and that presented in this paper. The effort to capture the input data for that approach was approximately 25 man-hours of effort [31]. This was considered worthwhile as drivers typically have a long lifetime and have to be certified regularly. We note how both COPs have the same set of constants, and are applied to FGMS I/O drivers having similar architectural design, and hence, the overhead for deriving the COP constant values is comparable in both cases. Furthermore, the design of our COP ensures that the final users, i.e., software engineers, can simply use it as a black box configuration generator, without having to be aware of the mathematical details of the COP. Currently, KM engineers spend several days simulating the FGMS behavior with manually tuned delay times, and monitoring its performance requirements. We expect that, by following our approach, they can configure the delay times in the FGMS drivers more conveniently, and ensure that no safety risks are posed by violating performance requirements at runtime.

Previous work in the field of software stress testing has shown that approaches based on complete search can potentially be more effective than metaheuristics in finding task schedules closer to the global optimum [14]. This aspect motivated us to investigate complete search strategies also in the context of performance tuning. However, we solve the COP with a off-the shelf solver that performs a deterministic complete search. This means that solving the COP multiple times within a time budget always finds the same set of solutions. To diversify the configurations found, we plan to combine complete deterministic search with randomized metaheuristics in hybrid strategies, which have already been successfully applied to stress test RTES [12].

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, LICS 1990, pp. 414–425. IEEE (1990)
2. Atamtürk, A., Savelsbergh, M.W.: Integer-programming software systems. Ann. Oper. Res. **140**(1), 67–124 (2005)
3. Baker, T.P.: An analysis of fixed-priority schedulability on a multiprocessor. Real-Time Syst. **32**(1–2), 49–71 (2006)
4. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems, vol. 39. Springer, New York (2001)
5. Beizer, B.: Software Testing Techniques. Dreamtech Press (2002)
6. Briand, L.C., Labiche, Y., Shousha, M.: Using genetic algorithms for early schedulability analysis and stress testing in real-time systems. Genet. Program. Evolvable Mach. **7**(2), 145–170 (2006)

7. Cambazard, H., Hladik, P.-E., Déplanche, A.-M., Jussien, N., Trinquet, Y.: Decomposition and learning for a hard real time task allocation problem. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 153–167. Springer, Heidelberg (2004)
8. Caseau, Y., Laburthe, F.: Improved CLP scheduling with task intervals. In: ICLP, pp. 369–383. Citeseer (1994)
9. Clarke, E.M., Klieber, W., Nováček, M., Zuliani, P.: Model checking and the state explosion problem. In: Meyer, B., Nordio, M. (eds.) LASER 2011. LNCS, vol. 7682, pp. 1–30. Springer, Heidelberg (2012)
10. David, A., Illum, J., Larsen, K., Skou, A.: Model-based framework for schedulability analysis using UPPAAL 4.1. In: Model-Based Design for Embedded Systems, p. 93 (2010)
11. Davis, R.I., Burns, A.: A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. (CSUR) **43**(4), 35 (2011)
12. Di Alesio, S., Briand, L., Nejati, S., Gotlieb, A.: Combining genetic algorithms and constraint programming to support stress testing of task deadlines. ACM Trans. Softw. Eng. Method. (2015)
13. Di Alesio, S., Gotlieb, A., Nejati, S., Briand, L.: Testing deadline misses for real-time systems using constraint optimization techniques. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), pp. 764–769. IEEE (2012)
14. Di Alesio, S., Nejati, S., Briand, L., Gotlieb, A.: Stress testing of task deadlines: a constraint programming approach. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 158–167. IEEE (2013)
15. Di Alesio, S., Nejati, S., Briand, L., Gotlieb, A.: Worst-case scheduling of software tasks. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 813–830. Springer, Heidelberg (2014)
16. Galorath, D.D., Evans, M.W.: Software Sizing, Estimation, And Risk Management: When Performance is Measured Performance Improves. CRC Press, Boca Raton (2006)
17. Gomaa, H.: Designing concurrent, distributed, and real-time applications with UML. In: Proceedings of the 28th International Conference on Software Engineering, pp. 1059–1060. ACM (2006)
18. Henzinger, T.A., Sifakis, J.: The embedded systems design challenge. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 1–15. Springer, Heidelberg (2006)
19. Hladik, P.E., Cambazard, H., Déplanche, A.M., Jussien, N.: Solving a real-time allocation problem with constraint programming. J. Syst. Softw. **81**(1), 132–149 (2008)
20. Huang, W., Lam, H.: Using genetic algorithms to optimize controller parameters for HVAC systems. Energy Build. **26**(3), 277–282 (1997)
21. Jaffar, J., Maher, M.J.: Constraint logic programming: a survey. J. Logic Program. **19**, 503–581 (1994)
22. Jain, R.: The Art of Computer Systems Performance Analysis. Wiley, New York (2008)
23. Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. Springer, New York (2011)
24. Laborie, P.: IBM ILOG CP optimizer for detailed scheduling illustrated on three problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 148–162. Springer, Heidelberg (2009)
25. Lala, J.H., Harper, R.E.: Architectural principles for safety-critical real-time applications. Proc. IEEE **82**(1), 25–40 (1994)

26. Le Pape, C., Baptiste, P.: An experimental comparison of constraint-based algorithms for the preemptive job shop scheduling problem. In: CP 1997 Workshop on Industrial Constraint-Directed Scheduling. Citeseer (1997)
27. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems: A Cyber-physical Systems Approach. Lee & Seshia (2011)
28. Malapert, A., Cambazard, H., Guéret, C., Jussien, N., Langevin, A., Rousseau, L.M.: An optimal constraint programming approach to the open-shop problem. INFORMS J. Comput. **24**(2), 228–244 (2012)
29. Nejati, S., Adedjouma, M., Briand, L.C., Hellebaut, J., Begey, J., Clement, Y.: Minimizing CPU time shortage risks in integrated embedded software. In: 2013 IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 529–539. IEEE (2013)
30. Nejati, S., Briand, L.C.: Identifying optimal trade-offs between CPU time usage and temporal constraints using search. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp. 351–361. ACM (2014)
31. Nejati, S., Di Alesio, S., Sabetzadeh, M., Briand, L.: Modeling and analysis of CPU usage in safety-critical embedded systems to support stress testing. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 759–775. Springer, Heidelberg (2012)
32. Reza Hejazi, S., Saghafian, S.: Flowshop-scheduling problems with makespan criterion: a review. Int. J. Prod. Res. **43**(14), 2895–2929 (2005)
33. Singh, A.: Identifying Malicious Code Through Reverse Engineering. Springer, New York (2009)
34. Sprunt, B., Sha, L., Lehoczky, J.: Aperiodic task scheduling for hard-real-time systems. Real-Time Syst. **1**(1), 27–60 (1989)
35. Storey, N.R.: Safety Critical Computer Systems. Addison-Wesley Longman Publishing Co., Inc., Boston (1996)
36. Tanenbaum, A.S.: Modern Operating Systems. Pearson Education (2009)
37. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocess. Microprogram. **40**(2), 117–134 (1994)
38. Van Hentenryck, P.: The OPL Optimization Programming Language. MIT Press, Cambridge (1999)
39. Varšek, A., Urbančič, T., Filipič, B.: Genetic algorithms in controller design and tuning. IEEE Trans. Syst. Man Cybern. **23**(5), 1330–1339 (1993)
40. Weyuker, E.J., Vokolos, F.I.: Experience with performance testing of software systems: issues, an approach, and case study. IEEE Trans. Softw. Eng. **26**(12), 1147–1156 (2000)
41. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., et al.: The worst-case execution-time problem - overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. (TECS) **7**(3), 36 (2008)
42. Woodside, M., Franks, G., Petriu, D.C.: The future of software performance engineering. In: Future of Software Engineering, FOSE 2007, pp. 171–187. IEEE (2007)
43. Yun, Y.S., Gen, M.: Advanced scheduling problem using constraint programming techniques in SCM environment. Comput. Ind. Eng. **43**(1), 213–229 (2002)

SABIO: An Implementation of MIP and CP for Interactive Soccer Queries

Robinson Duque[1(✉)], Juan Francisco Díaz[1], and Alejandro Arbelaez[2]

[1] Universidad del Valle, Cali, Colombia
{robinson.duque,juanfco.diaz}@correounivalle.edu.co
[2] Insight Centre for Data Analytics, University College Cork, Cork, Ireland
alejandro.arbelaez@insight-centre.org

Abstract. Soccer is one of the most popular sports in the world with millions of fans that usually raise interesting questions when the competition is partially completed. One interesting question relates to the *elimination problem* which consists in checking at some stage of the competition if a team i still has a theoretical chance to become the champion. Some other interesting problems from literature are the *guaranteed qualification problem,* the *possible qualification problem, the score vector problem, promotion and relegation.* These problems are NP-complete for the actual FIFA pointing rule system (0 points-loss, 1 point-tie, 3 points-win). SABIO is an online platform that helps users discover information related to soccer by letting them formulate questions in form of constraints and go beyond the classical soccer computational problems. In the paper we considerably improve the performance of an existing CP model and combine the use of MIP and CP to answer general soccer queries in a real-time application.

1 Introduction

A soccer competition (league or tournament) consists of n teams playing against each other in a single or double round-robin schedule. Tournament competitions are usually played in two-stages: a single or double round-robin schedule for the regular season and a final knockout stage (aka playoffs) where typically eight teams qualify. On the other hand, league competitions consist of a single-stage where each team i gets to play against team j once or twice and the first team in the standing table becomes the champion.

The elimination problem is well-known in sports competitions [1,2] and consists in determining whether at some stage of the competition a given team still has the opportunity to be within the top teams to qualify for playoffs or become the champion. This problem was proved NP-Complete for the current FiFA score system (0 points-loss, 1 point-tie, 3 points-win) [3,4]. However, in [4] the authors pointed out that with the old FIFA score system (0, 1, 2) from the

90's, the elimination problem could be solved in polynomial time using network flow algorithms as first proposed in [5]. Kern and Paulusma [3,6] generalized NP-completeness depending on the sports' score system and showed that questions like "is there a chance that team i ends up being one of the three teams that have the three lowest final scores?", is actually an NP-complete problem. Later, Pálvölgyi showed that deciding whether a given score vector is a possible outcome of a soccer-tournament (i.e., *score vector problem*) is also NP-complete [7].

Recently, [8] proposed a MIP formulation to tackle the *guaranteed qualification problem* to find the minimum number of points a given team has to win to become champion or qualify to the playoffs. However, this model is limited to single round-robin competitions. In this paper, we use some ideas from [8], and we extend it for single and double round-robin competitions. Additionally, we would like to point out that the flexibility of our model allows us to answer more queries, i.e., *game result queries*, *position in ranking queries*, *relative position queries*, and *final position queries*, that let users create different scenarios and go beyond the classical soccer computation problems.

SABIO (Soccer Analysis Based on Inference Outputs) is an online platform, available at www.sabiofutbol.com, capable of answering soccer related queries by letting users formulate questions in form of constraints. In this paper we considerably improve an existing CP model [9], and we combine the use of MIP and CP to improve the performance of SABIO.

2   CP Model for Interactive Soccer Queries

Constraint programming (CP) is a powerful paradigm that can be used to solve combinatorial problems. Typically, CP combines backtracking search with constraint propagation to filter inconsistent values and reduce the search space. In [9], we described a CP model for position in ranking queries. Here we extend such model by introducing three new type of queries, i.e., *game results*, *relative position*, *final points*. Additionally, we also propose a set of redundant constraints that help to prune the search tree for position ranking queries.

Basic Soccer Model: these variables capture basic information to formulate a model for soccer competitions.

- n: number of teams in the competition;
- T: set of team indexes in the competition;
- i, j: team indexes, such that $(i, j \in T)$;
- p_i: initial points of team i. If i has not played any games, then $p_i = 0$;
- F: number of fixtures left to be played in the competition. A fixture consists of one or more games between competitors;
- k: represents a fixture number, $(1 \leq k \leq F)$;
- G: set that represents the schedule of the remaining games to be played. Every game is represented as a triple $ng_e = (i, j, k) \wedge 0 \leq e \leq |G|$, where k is the fixture when both teams meet in a game;
- pt_{ik}: represents the points that team i gets in fixture k, $(1 \leq k \leq F$ and $pt_{ik} \in \{0, 1, 3\})$. If team i is not scheduled to play fixture k, then $pt_{i,k} = 0$.

- tp_i: total points of team i at end of the competition;
- geq_{ij}: Boolean variable indicating if team j has greater or equal total points as i: if $tp_j \geq tp_i$ then $geq_{ij} = 1$; otherwise $geq_{ij} = 0$ $(\forall i, j \in T)$;
- eq_{ij}: boolean variable indicating if two different teams i and j tie in points at the end of the competition: if $tp_j = tp_i$ and $i \neq j$ then $eq_{ij} = 1$; otherwise $eq_{ij} = 0$ $(\forall i, j \in T)$.
- pos_i: position of team i at the end of the competition;
- $worstPos_i$: upper bound for pos_i;
- $bestPos_i$: lower bound for pos_i;

Constraint (1) represents a valid game point assignment (0,3), (3,0) or (1,1) for each game $ng_e \in G$ between two teams i and j in a fixture k and constraint (2) corresponds to the final points tp_i of a team i:

$$2 \leq pt_{ik} + pt_{jk} \leq 3 \quad \forall ng_e \in G \wedge ng_e = (i, j, k) \tag{1}$$

$$tp_i = p_i + \sum_{k=1}^{F} pt_{ik} \quad \forall i \in T \tag{2}$$

Constraints (3) to (6) are used to calculate final positions. All the final positions must be different and every position is bounded by $bestPos_i$ and $worstPos_i$:

$$worstPos_i = \sum_{j=1}^{n} geq_{ij} \quad \forall i, j \in T \tag{3}$$

$$bestPos_i = worstPos_i - \sum_{j=1, j \neq i}^{n} eq_{ij} \quad \forall i, j \in T \wedge i \neq j \tag{4}$$

$$bestPos_i \leq pos_i \leq worstPos_i \quad \forall i \in T \tag{5}$$

$$alldifferent(pos_1, ..., pos_n) \tag{6}$$

Game Result Queries: Constraint (7) allows users to include assumptions about the outcome of remaining games to constrain the points of teams i and j in a fixture k, e.g., Barcelona ends in a tie with R. Madrid:

- Q: set of game result queries for a pair of teams (i, j) in a fixture k. Every query is defined as a tuple $nq_a = (ptc_{ik}, ptc_{jk})$ and $0 \leq a \leq |Q|$;
- ptc_{ik} and ptc_{jk}: are user suppositions about the points that a pair of teams (i, j) will get in a fixture k, i.e.,$(ptc_{ik}, ptc_{jk}) \in \{(0, 3), (3, 0), (1, 1)\}$);

$$(pt_{ik} = ptc_{ik} \wedge pt_{jk} = ptc_{jk}) \quad \forall nq_a \in Q \wedge nq_a = (ptc_{ik}, ptc_{jk}) \tag{7}$$

Position in Ranking Queries: we use this set of constraints to indicate whether a given team can be above, below, or at a given position ptn_i, e.g., R.Madrid will be in position 3. Constraint (8) depicts three of the five possibilities:

- P: set of possible position in ranking queries, defined as a set of triples $np_b = (i, opr_i, ptn_i)$ and $0 \leq b \leq |P|$;

- opr_i: logical operator ($opr_i \in \{<, \leq, >, \geq, =\}$) to constrain team i;
- ptn_i: denoting the expected position for team i; $1 \leq ptn_i \leq n$;

$$\forall np_b \in P \land np_b = (i, opr_i, ptn_i) \begin{cases} pos_i = ptn_i, & \text{if } opr_i \text{ is } = \\ pos_i < ptn_i, & \text{if } opr_i \text{ is } < \\ pos_i > ptn_i, & \text{if } opr_i \text{ is } > \end{cases} \qquad (8)$$

Relative Position Queries: these queries indicate whether a given team i will be above, below, or equal to another team j at the end of the tournament and constraint (9) depicts three of the five queries, e.g., Barcelona will be in a better position than R. Madrid. In this particular case we use tp_i and tp_j instead of pos_i and pos_j. We consider that two teams i and j might tie up in the same position if they have the same points at the end of the competition. We recall that we do not use pos_i and pos_j due to the *alldifferent* constraint in (6).

- R: set of possible relative position queries defined as a set of triples $nr_c = (i, op_{ij}, j)$ and $0 \leq c \leq |R|$;
- op_{ij}: denoting a logical operator ($op_{ij} \in \{<, \leq, >, \geq, =\}$) to constrain a pair of teams i and j.

$$\forall nr_c \in R \land nr_c = (i, op_{ij}, j) \begin{cases} tp_i = tp_j, & \text{if } op_{ij} \text{ is } = \\ tp_i < tp_j, & \text{if } op_{ij} \text{ is } < \\ tp_i > tp_j, & \text{if } op_{ij} \text{ is } > \end{cases} \qquad (9)$$

Final Point Queries: (also known as score queries) we use these variables for queries about the final points of the teams, e.g., Barcelona scores at the end of the competition 75 points. Constraint (10) guarantees *final point queries*.

- S: set of possible final point queries defined as a set of tuples $ns_d = (i, s_i)$ and $0 \leq d \leq |S|$;
- s_i: denoting the wanted final points of team i.

$$(tp_i = s_i) \quad \forall ns_d \in S \land ns_d = (i, s_i) \qquad (10)$$

3   Extended CP Model

In our CP model, the position bounds (i.e., $bestPos_i$ and $worstPos_i$) for *position in ranking queries* can only be computed after finding the total points (tp_i) for all the teams in the competition, then the position constraints are validated. This formulation leads to an exhaustive search with a late pruning rule based on the teams positions.

The redundant constraints proposed in this section make inferences about the teams positions based on the total points, in order to start pruning as early as possible while the search unfolds. To depict our approach, consider the following

position in ranking constraint: *"A will be in the same position as 1"* which can be represented as the triplet $(A, =, 1)$ or $pos_A = 1$ according to Constraint (8). In order to satisfy such constraint, *it must hold that during the search, the number of teams with more points than A has to be 0*, otherwise, A will never be in first position. To take this kind of scenario into account we propose a set of redundant constraints that constantly validate the number of teams with more (resp. less) points than A. Therefore, let L denote the set of constrained teams included in all the triples $np_b \in P$, such that $np_b = (i, opr_i, ptn_i)$ where $i \in L$ and $L \subseteq T$. Now, let us start by introducing a set of variables for constrained teams $i \in L$:

– $less_{ij}$: Boolean variables denoting whether teams j have less points than i, i.e., if $tp_j < tp_i$ then $less_{ij} = 1$; otherwise $less_{ij} = 0$ $(\forall j \in T \wedge \forall i \in L)$;
– $grtr_{ij}$: Boolean variables denoting whether teams j have more points than i, i.e., if $tp_j > tp_i$ then $grtr_{ij} = 1$; otherwise $grtr_{ij} = 0$ $(\forall j \in T \wedge \forall i \in L)$.

"Greater than" redundant constraint, i.e., $np_b = (i, >, ptn_i)$. During search, the number of teams with fewer points than team i must be limited to $(n\text{-}ptn_i)$:

$$\sum_{j=1, j \neq i}^{n} less_{ij} < (n - ptn_i) \quad \forall j \in T \wedge \forall i \in L \tag{11}$$

Similarly, we use $\sum_{j=1, j \neq i}^{n} less_{ij} \leq (n - ptn_i)$ for constraints $np_b = (i, \geq, ptn_i)$.

"Less than" redundant constraint, i.e., $np_b = (i, <, ptn_i)$. During search, the number of teams with more points than team i must be limited to $(ptn_i\text{-}1)$:

$$\sum_{j=1, j \neq i}^{n} grtr_{ij} < (ptn_i - 1) \quad \forall j \in T \wedge \forall i \in L \tag{12}$$

Similarly, we use $\sum_{j=1, j \neq i}^{n} grtr_{ij} \leq (ptn_i - 1)$ for constraints $np_b = (i, \leq, ptn_i)$.

"Equal to" redundant constraint, i.e., $np_b = (i, =, ptn_i)$. During search, we constrain the number of teams above (resp. below) of a team i to:

$$\sum_{j=1, j \neq i}^{n} grtr_{ij} < (ptn_i) \quad \forall j \in T \wedge \forall i \in L \tag{13}$$

$$\sum_{j=1, j \neq i}^{n} less_{ij} < (n - ptn_i + 1) \quad \forall j \in T \wedge \forall i \in L \tag{14}$$

Variable/Value Selection: In SABIO the variable/value selection strategies involve identifying the outcome of a game for a selected team. Generic heuristics (e.g., [10,11]) typically do not perform well in this domain as they do not exploit the structure of the problem. Therefore, in [9] we proposed a variable selection system of priorities for *position ranking* queries. We use nine strategies $\{S1 \ldots S9\}$ for the outcome of the game in the value selection process. Each strategy represents the probability of a win, tie, or lose. For instance, $S9 = \langle 0.25,$

0.25, 0.5⟩, indicates that the selected team wins or ties the game with a probability of 0.25 each and loses with a probability of 0.5. We use decision trees to select the most suitable strategy for teams constrained with the equal operator.

For *relative position queries* we use an alternative approach. Let us assume we want to answer a *greater than* query for two teams (i.e., $tp_i > tp_j$). In this case it is natural to get team i to *win* and team j to *lose* the remaining games. Therefore, we increase the priority of both teams to be selected during branching (similarly for less than queries). Alternatively, for queries indicating $(tp_i = tp_j)$ we found that both teams have to be assigned a high priority and also that a (win, lose) strategy may generate final points overshooting, therefore, we decided to assign both teams a *tie* strategy.

For *final point queries* we assign a high priority to teams involved in at least one query. Finally, *game result queries* can be trivially solved with our models by only using constraint propagation.

Sequential and Parallel Restart-Based Search: Inspired by the quickest first principle [12], we execute the strategies in a predefined order and we use a restart-based search with a fixed time cutoff. For teams involved in at least one query we use the above mentioned variable/value selection heuristics in all restarts. For the remaining teams, we use a different strategy for each restart, starting with S1 for the first restart and using S9 for the ninth restart, in the tenth restart we use a random strategy for unconstrained teams. The last restart is executed until a solution is observed or a time limit is reached.

These restart strategies can be executed either *sequentially* or in *parallel* for a fixed cutoff time (i.e., one restart after another in a single core, or one restart per core in a multi-core machine). In the parallel version with four cores, we execute in parallel (for unconstrained teams) {S1, ..., $S4$} followed by {S5, ..., $S9$}. We finish the execution with the random strategy for all cores.

4   Empirical Evaluation

Tests Configuration: We evaluated our models using CPLEX (V12.6.2) as our MIP reference solver and Mozart-Oz (V 1.4.0) as our CP reference solver. All the experiments were performed in a 4-core machine featuring an Intel Core i5 processor at 2.3 Ghz and 4 GB of RAM. In particular we focus our attention in the Colombian league (liga Postobón 2014-I) with 18 teams and 18 fixtures to play in a single round-robin schedule (17 fixtures + 1 extra fixture for the derbies). We provided five experimental scenarios by exploring different stages of the competition (i.e., fixtures 7, 9, 11, 14, and 16). For each fixture we created instances with position in ranking queries (P), relative position queries (R), and final point queries (S). We excluded game results queries (Q) from our experiments as they can be trivially solved with our models.

The scenarios for every query type (P, R, S) and fixture (7, 9, 11, 14, 16) included 100 instances with 2 suppositions, 100 with 3 suppositions, and the same for 4, 5, 7, and 9 suppositions. For each instance (9000 in total) we used a time limit of 30 s. We recall that our models are implemented in SABIO, a
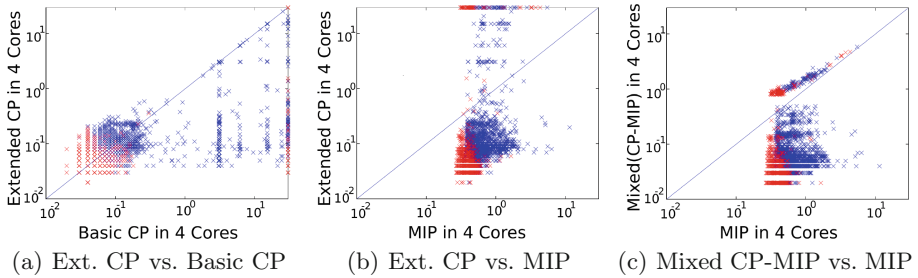
(a) Ext. CP vs. Basic CP   (b) Ext. CP vs. MIP   (c) Mixed CP-MIP vs. MIP

Fig. 1. Runtime (in secs) comparison of the different approaches (Color figure online)

Web based application and long answer times are not desirable. In order to analyze the performance of the models, we reported executions with 1 and 4 cores and experimented four scenarios separately: the *basic CP model* proposed in [9]; the *extended CP model* with redundant constraints proposed in this paper; an extended version of the *MIP model* of [8] able to deal with the same queries as our CP model; and a *mixed* execution of our CP and MIP models (i.e., we observed in our experiments that MIP is slightly better for a small number of instances (mainly unsat). Therefore, to exploit the best features of both CP and MIP, we run first our CP model during 0.5 s (average time in 1 core) to solve as many instances as possible, then run our MIP implementation for 29.5 s).

Tests Results: We start with Fig. 1 where we compare the performance of extended model with the redundant constraints against the basic model from [9], the MIP model, and the mixed approach CP-MIP. The x-axis gives the runtime of the extended model and the y-axis gives the runtime of the basic model (resp. MIP model). Blue (resp. red) points indicate SAT (resp. UNSAT). Points below the diagonal indicate that the extend model is faster. In Fig. 1(a) it can be observed that the extended version is typically better (w.r.t. speed and capacity solving) than the basic model, except for a few instances where the runtime for both approaches is less than 0.1 s. Figure 1(b) shows that the extended model is considerably faster than the MIP model on 8744 instances. Interestingly MIP is particularly better than CP for UNSAT instances, in fact, 72 % of the unsolved instances for the CP model are UNSAT. We attribute this to the fact that for UNSAT instances it is necessary to explore the complete search tree. In Fig. 1(c) we exploit the complementary behaviour of MIP and our extended CP model, here we observe that only for 243 (out of 9000) instances it would have been better to alternate the execution of MIP and CP.

We now move our attention to Table 1 where we present complete statistics of the four approaches. This table shows the number of unsolved instances (top), the standard deviation and the average run-times (bottom) for solved instances. Certainly, R and S queries are the easiest ones and nearly all instances can be solved within the time limit. However, we observed that for these instances the extended CP model is faster than MIP. Alternatively, P queries are the hardest ones, particularly for our CP approaches.

Table 1. Unsolved instances and run-times (avg, std) in seconds

Num. cores	1 Core			4 Cores		
Query type	P Queries	R Queries	S Queries	P Queries	R Queries	S Queries
Unsolved B. CP	627	-	1	607	-	1
Unsolved E. CP	171	-	1	166	-	1
Unsolved MIP	5	-	-	1	-	-
Uns. mixed (CP-MIP)	2	-	-	1	-	-
Avg/std (B. CP)	1.60/3.97	0.05/0.02	0.06/0.24	0.83/2.97	0.05/0.02	0.05/0.04
Avg/std (E. CP)	0.55/2.08	0.05/0.02	0.06/0.24	0.31/1.66	0.05/0.02	0.05/0.05
Avg/std (MIP)	0.54/0.47	0.41/0.09	0.40/0.09	0.58/0.46	0.40/0.06	0.41/0.07
Avg/std mixed(CP-MIP)	**0.19**/0.58	0.05/0.01	0.06/0.03	**0.16**/0.39	**0.04**/0.02	**0.04**/0.03

We would like to highlight the importance of the redundant constraints in the extended CP model. The new constraints help to solve 456 more instances than the basic model from [9] and it also considerably improves the average runtime from 0.83 to 0.31 s for 4 cores.

Additionally, we observe that the MIP and the extended CP approach are complementary. The MIP model in a 4-core execution, is able to solve more instances, i.e., 607 more than the basic CP approach and 166 more than the extended CP with redundant constraints. The extended CP approach is about 46 %, 87 %, and 88 % faster than the MIP approach for P, R, and S queries using 4 cores. Interestingly, for P queries we observe that extended CP in 1 and 4 cores present a less uniform behaviour (std: 2.08 &1.66) compared to MIP (std: 0.47 &0.46). We attribute this to answers found in later restarts. Such behaviour can also be observed in Fig. 1(b), where most of MIP instances range between 0.1 and 10 secs with observed min. (resp. max.) runtime values of 0.28 (resp. 1.46) s, while extended CP ranges between 0.01 and 100 with min. (resp. max.) values of 0.03 (resp. 25.38) s.

Finally, the overall best approach is the mixed solution between CP and MIP, this solution exploits the best features of both alternatives. We would like to remark that SABIO has been highlighted as an outstanding platform for soccer fans in Colombian's main news paper such as *El Tiempo* and *ADN Cali*. We are planning a new release of SABIO with the models described in this paper.

5   Conclusions

In this paper we have improved an existing CP model to solve general soccer fan queries at different stages of the computation. We compared our improved CP approach against a MIP formulation and observed a complementary behaviour between the two approaches. The CP approach is considerably faster than the MIP model. However, the MIP model is able to solve more instances than the

CP one. We expect our SABIO Web application to interact with thousands of users at the same time, therefore both speed and capacity solving of the two models are expected to play an important role for a fast and robust solution.

References

1. Schwartz, B.L.: Possible winners in partially completed tournaments. SIAM Rev. **8**(3), 302–308 (1966)
2. Hoffman, A., Rivlin, T.: When is a team "mathematically" eliminated? In: Princeton Symposium on Mathematical Programming, pp. 391–401. Princeton, NJ (1967)
3. Kern, W., Paulusma, D.: The new FIFA rules are hard: complexity aspects of sports competitions. Discrete Appl. Math. **108**(3), 317–323 (2001)
4. Bernholt, T., Gälich, A., Hofmeister, T., Schmitt, N.: Football elimination is hard to decide under the 3-point-rule. In: MFCS, pp. 410–418 (1999)
5. Wayne, K.D.: A new property and a faster algorithm for baseball elimination. SIAM J. Discrete Math. **14**(2), 223–229 (2001)
6. Kern, W., Paulusma, D.: The computational complexity of the elimination problem in generalized sports competitions. Discrete Optim. **1**(2), 205–214 (2004)
7. Pálvölgyi, D.: Deciding soccer scores and partial orientations of graphs. Acta Univ. Sapientiae **1**(1), 35–42 (2009)
8. Ribeiro, C.C., Urrutia, S.: An application of integer programming to playoff elimination in football championships. Int. Trans. Oper. Res. **12**(4), 375–386 (2005)
9. Duque, R., Díaz, J.F., Arbelaez, A.: Constraint programming and machine learning for interactive soccer analysis. In: LION 10 (2016, to appear)
10. Arbelaez, A., Hamadi, Y.: Exploiting weak dependencies in tree-based search. In: SAC 2009, pp. 1385–1391 (2009)
11. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. In: IJCAI 1979, San Francisco, CA, USA, pp. 356–364 (1979)
12. Borrett, J., Tsang, E.P., Walsh, N.R.: Adaptive constraint satisfaction: the quickest first principle. In: European Conference on Artificial Intelligence (1996)

Constraint Programming Models
for Chosen Key Differential Cryptanalysis

David Gerault[4(✉)], Marine Minier[1,2], and Christine Solnon[1,3]

[1] Université de Lyon, INSA-Lyon, 69621 Villeurbanne, France
{marine.minier,christine.solnon}@insa-lyon.fr
[2] CITI, INRIA, Villeurbanne, France
[3] LIRIS, CNRS UMR5205, Villeurbanne, France
[4] LIMOS, Clermont-ferrand, France
dagerault@gmail.com

Abstract. In this paper, we introduce Constraint Programming (CP) models to solve a cryptanalytic problem: the chosen key differential attack against the standard block cipher AES. The problem is solved in two steps: In Step 1, bytes are abstracted by binary values; In Step 2, byte values are searched. We introduce two CP models for Step 1: Model 1 is derived from AES rules in a straightforward way; Model 2 contains new constraints that remove invalid solutions filtered out in Step 2. We also introduce a CP model for Step 2. We evaluate scale-up properties of two classical CP solvers (Gecode and Choco) and a hybrid SAT/CP solver (Chuffed). We show that Model 2 is much more efficient than Model 1, and that Chuffed is faster than Choco which is faster than Gecode on the hardest instances of this problem. Furthermore, we prove that a solution claimed to be optimal in two recent cryptanalysis papers is not optimal by providing a better solution.

1 Introduction

Cryptography ensures properties such as confidentiality, integrity and signature of communications. Cryptanalysis aims at testing whether these properties are actually guaranteed. Whereas public key cryptography relies on hard problems, symmetric key cryptography relies on simple operations that are iterated many times to speed up encryption/decryption. The most important symmetric key primitives are hash functions and ciphers.

Hash functions guarantee integrity by creating a fixed size fingerprint of messages. Many cryptanalytic results have completely broken the standards MD5, SHA-0 and SHA-1 [11,23,24] by finding collisions, *i.e.*, messages with a same fingerprint. SAT solvers have been used to find collisions [6,12,15] and also against the future hash standard Keccak [16].

Ciphers guarantee confidentiality by encoding the original message into a different message, using a key, in such a way that the encoded message can further be decoded into the original one. Stream ciphers encode streams "on the fly", whereas block ciphers split the text in blocks which are encoded separately. Different approaches have been proposed for applying CP to cryptanalysing stream ciphers: [20] proposes to solve algebraic systems of equations that link together keys and encoded streams; [17] uses mixed integer linear programming to compute bounds for the Enocove-128v2 stream cipher. Since the seminal works of [3,9], several results appeared on block cipher cryptanalysis [17,21,22], mostly based on Mixed-Integer Programming.

Overview of the paper. In this paper, we focus on the cryptanalysis against block ciphers proposed in [3,9] and described in Sect. 2. The problem is usually solved in 2 steps: In Step 1, bytes are abstracted by binary values; In Step 2, byte values are searched. In Sect. 3, we describe a first CP model for Step 1, initially proposed in [14]. This model generates many invalid solutions that are filtered out in Step 2 (as initially proposed in [3,9]). In Sect. 4, we introduce new constraints that remove most of these invalid solutions. In Sect. 5, we briefly describe a CP model for Step 2. In Sect. 6, we evaluate scale-up properties of two classical CP solvers (Choco and Gecode) and a hybrid CP/SAT solver (Chuffed). We show that the new model for Step 1 is much more efficient than the initial model, and that Chuffed is faster than Choco which is faster than Gecode on the hardest instances of the problem. Furthermore, we prove that a solution claimed to be optimal in [3,9] is not optimal by providing a better solution. Actually, CP allows us not only to solve cryptanalysis problems more efficiently than the dedicated approaches of [3,9], but also in a safer way as it is easier to check the correctness of a CP model than the correctness of a dedicated program.

2   Problem Statement

In this Section, we detail the general structure of the AES (Advanced Encryption Standard) block cipher [8]. We then describe what a differential attack is and finally introduce the chosen key differential attack model.

2.1   AES Block Cipher

A block cipher is a function $E : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^n$ which, given a binary block X (called plaintext) of length n and a binary key K of length l, outputs a binary ciphered text $E(X, K)$ of length n such that $X = E^{-1}(E(X, K), K)$.

Most of today's block ciphers have an iterated structure: They apply a round function f r times so that $E(X, K) = X_r$ with $X_0 = X$ and $X_{i+1} = f(X_i, K_{i+1})$ for all $i \in [0; r-1]$.

Two famous examples of block ciphers are DES (Data Encryption Standard), which was the encryption standard between 1977 and 2000, and AES [8] which is the actual standard since 2001. AES ciphers blocks of length $n = 128$ bits,
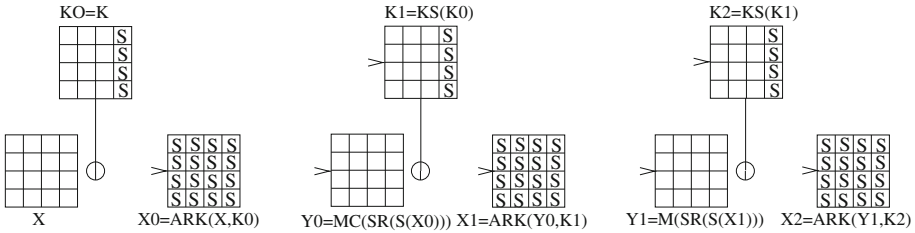
Fig. 1. AES ciphering process with $r = 2$ rounds. Each 4×4 array represents a group of 16 bytes. Before the first round, X_0 is obtained by applying ARK on the initial text X and the initial key $K = K_0$. Then, for each round $i \in [1, 2]$, S, SR and MC are applied on X_i to obtain Y_i, KS is applied on K_{i-1} to obtain K_i, and ARK is applied on K_i and Y_{i-1} to obtain X_i. Bytes that pass through the S-box are signaled by an S.

where each block is seen as a 4×4 matrix of bytes, where a byte is a sequence of 8 bits.

Given a 4×4 matrix of bytes M, we note $M[j]$ the 4 bytes at column $j \in [0, 3]$, and $M[j][k]$ the byte at column $j \in [0, 3]$ and row $k \in [0, 3]$.

The length of keys is $l \in \{128, 192, 256\}$. The number of rounds depends on the key length: $r = 10$ (resp. 12 and 14) for $l = 128$ (resp. 192 and 256). In this Section, we describe AES for $l = 128$.

The round function f of AES uses an SPN (Substitution-Permutation Network) structure and is described in Fig. 1 for $r = 2$ rounds. Before the first round, AddRoundKey is applied on the original plaintext X and the initial key $K_0 = K$ to obtain $X_0 = ARK(X, K_0)$. Then, for each round $i \in [0, r - 1]$:

– SubBytes, ShiftRows and MixColumns are applied on X_i to obtain
  $Y_i = MC(SR(S(X_i)))$,
– KeySchedule is applied on K_i to obtain $K_{i+1} = KS(K_i)$,
– AddRoundKey is applied on Y_i and K_{i+1} to obtain $X_{i+1} = ARK(Y_i, K_{i+1})$.

These different operations are described below.

SubBytes S. The S operation, also called S-box, is a non-linear permutation which is applied on each byte of X_i separately, *i.e.*, for each $j, k \in [0, 3]$, S substitutes $X_i[j][k]$ by $S(X_i[j][k])$, according to a lookup table.

ShiftRows SR. SR is a linear mapping that rotates on the left by one byte position (resp. 2 and 3 byte positions) the second row (resp. third and fourth rows) of the current matrix $S(X_i)$, *i.e.*, for each $j, k \in [0, 3]$:

$$SR(S(X_i))[j][k] = S(X_i)[(k + j)\%4][k]$$

MixColumns MC. MC is a linear mapping that multiplies each column of the input matrix $SR(S(X_i))$ by a 4×4 fixed matrix chosen for its good properties of

diffusion (see [5]). In particular, it has the Maximum Distance Separable (MDS) property: For each column $j \in [0,3]$, it ensures:

$$w(SR(S(X_i))[j]) + w(MC(SR(S(X_i)))[j]) \in \{0,5,6,7,8\}$$

where w is a function which returns the number of bytes different from 0^8 (we note 0^8 the byte composed of 8 bits equal to 0).

AddRoundKey ARK. ARK performs a xor operation (noted $\oplus$) between Y_i and subkey K_{i+1}, *i.e.*, for each column and row $j,k \in [0,3]$,

$$ARK(Y_i[j][k], K_{i+1}[j][k]) = Y_i[j][k] \oplus K_{i+1}[j][k]$$

KeySchedule KS. The subkey at round 0 is the initial key, *i.e.*, $K_0 = K$. For each round $i \in [1,r]$, the subkey K_i is generated from the previous subkey K_{i-1} by applying the key schedule, *i.e.*, $K_i = KS(K_{i-1})$. For keys of length $l = 128$ bits, each subkey K_i is a 4×4 byte matrix. KS operates on columns:

– It first computes the first column $K_i[0]$ from K_{i-1} as follows:

$$\forall k \in [0;3], K_i[0][k] = K_{i-1}[0][k] \oplus S(K_{i-1}[3][(k+1)\%4])$$

where S is the SubBytes operation. Moreover, $r+1$ predefined constants are added to $K_i[0][0]$.
– For the last 3 columns $j \in \{1,2,3\}$, we have: $K_i[j] = K_i[j-1] \oplus K_{i-1}[j]$

2.2 Differential Cryptanalysis

Differential cryptanalysis was introduced in 1991 [2], and aims at evaluating confidentiality by testing whether it is possible to find the secret key within a reasonable number of trials. The idea is to consider plaintext pairs (X, X') and to study the propagation of the initial difference between X and X' while going through the successive rounds. We note δX_i the xor difference between the two plaintexts X_i and X_i' obtained after the ith round of the ciphering of X and X', *i.e.*, $\delta X_i = X_i \oplus X_i'$, and we say that $\delta X_i[j][k] = X_i[j][k] \oplus X_i'[j][k]$ is a *differential byte* (for each column and row $j,k \in [0,3]$).

Let us keep in mind that the round function f is composed of a set of linear operations (SR, MC, ARK) and a non linear operation (S). The linear operations only move differences to other places. Indeed, for every linear operation $l \in \{SR, MC, ARK\}$, we have $l(A \oplus B) = l(A) \oplus l(B)$. So, we can easily predict how differences are propagated from δX_i to δX_{i+1} by these operations.

The non linear operation S has to be studied more carefully. As said before, S operates on each byte $X_i[j][k]$ separately. Therefore, we need to study how the S-box propagates differences for a pair (A, B) of bytes. To this aim, we evaluate the probability that the output difference $S(A) \oplus S(B)$ is equal to β when the input difference $A \oplus B$ is equal to α, where α and β are bytes. This probability is denoted $D_{\alpha,\beta}$ and is defined by

$$D_{\alpha,\beta} = \frac{\#\{(A,B) \in \{0,1\}^8 \times \{0,1\}^8 \mid (A \oplus B = \alpha) \wedge (S(A) \oplus S(B) = \beta)\}}{256}$$

For example, let us consider an input difference $\alpha = 00000001$ and an output difference $\beta = 00100000$. For the AES S-box, the transition from 00000001 to 00100000 only occurs for 4 couples of inputs, among the 256 possible couples so that $D_{00000001,00100000} = \frac{4}{256}$. For the AES S-box, most of the times the transition probability is equal to $\frac{0}{256}$ or $\frac{2}{256}$, and rarely to $\frac{4}{256}$. Note that S is a bijection so that $A \oplus B = 0^8 \Leftrightarrow S(A) \oplus S(B) = 0^8$. As a consequence, $D_{0^8,0^8} = 1$. In other words, if there is no difference in the input $A \oplus B$, then there is no difference in the output $S(A) \oplus S(B)$.

Then, for each round $i \in [0; r]$, we study difference propagation when the 16 bytes of X_i and X_i' pass through the S-box. For each column $j \in [0; 3]$ and each row $k \in [0; 3]$, we note $\delta X_i[j][k] = X_i[j][k] \oplus X_i'[j][k]$ and $\delta S X_i[j][k] = S(X_i[j][k]) \oplus S(X_i'[j][k])$ the difference for the byte at column j and row k before and after passing the S-box, respectively. The probability of obtaining the output difference $\delta S X_i[j][k]$ when the input difference is $\delta X_i[j][k]$ is given by $D_{\delta X_i[j][k],\delta S X_i[j][k]}$. Hence, the probability of obtaining the output difference $\delta X_r = X_r \oplus X_r'$ after r rounds given an input difference $\delta X = X \oplus X'$ is:

$$p_1(\delta X_r | \delta X) = \prod_{i=0}^{r} \prod_{j=0}^{3} \prod_{k=0}^{3} D_{\delta X_i[j][k],\delta S X_i[j][k]} \qquad (1)$$

We refer the reader to [2] for more details.

A first goal of the attacker is to find the values of δX_i for $i \in \{0, \ldots r\}$ which maximize the probability p_1. Once done, the attacker retrieves some partial information on the secret key K from the next subkey $K_{r+1} = KS(K_r)$. To do so, the attacker has to cipher M chosen plaintext pairs (X, X') to obtain M ciphered pairs (C, C'). From those pairs (C, C'), the attacker deciphers the last round to partially retrieve δX_r according to all possible values of some bits of K_{r+1}. The correct key will be the one for which the optimal value of δX_r (that maximizes p_1) appears the most frequently. The number M of plaintext pairs required for the success of the attack may be directly computed from p_1 and is equal to c/p_1 for c a small constant as shown in [2].

2.3   Chosen Key Differential Cryptanalysis

Today, differential cryptanalysis is public knowledge, so modern block ciphers such as AES have been designed to have proven bounds against differential attacks. However, in 1993, E. Biham proposed a new type of attack called related key attack [1] that allows an attacker to inject differences not only between the plaintexts X and X' but also between the keys K and K' (even if the secret key K stays unknown from the attacker) to try to mount more powerful attacks. The goal of the attack stays the same as previously, *i.e.*, try to find some partial information on the secret key K by testing some bits of the last subkey K_r on the veracity of the differential relation which happens with probability p_1.

We note $K_i[j][k]$ and $K_i'[j][k]$ the bytes at column j and row k in the subkeys of K and K' at round i, $\delta K_i[j][k]$ the difference between $K_i[j][k]$ and $K_i'[j][k]$, i.e.,

$\delta K_i[j][k] = K_i[j][k] \oplus K'_i[j][k]$, and $\delta SK_i[j][k]$ the difference between $S(K_i[j][k])$ and $S(K'_i[j][k])$, i.e., $\delta SK_i[j][k] = S(K_i[j][k]) \oplus S(K'_i[j][k])$. As the only bytes of the subkeys that pass through the S-box are those that are at column $j = 3$, Eq. (1) is modified by multiplying it by $D_{\delta K_i[3][k], \delta SK_i[3][k]}$, for each round i and each line k, i.e., the goal of the attacker is to find the values of δX_i and δK_i for $i \in \{0, \ldots r\}$ which maximize the probability p_2 defined by Eq. (2).

$$p_2(\delta X_r, \delta K_r | \delta X, \delta K) = p_1(\delta X_r | \delta X) * \prod_{i=0}^{r} \prod_{k=0}^{3} D_{\delta K_i[3][k], \delta SK_i[3][k]} \qquad (2)$$

2.4   Two Step Solving Process for Chosen Key Differential Cryptanalysis

Two main papers [3,9] describe results for the chosen key differential cryptanalysis of the AES and propose algorithms for finding initial pairs of plain texts and keys which maximize the probability p_2. In both papers, the problem is solved in two steps.

First step: Search of binary solutions. In the first step, each unknown δX_i is modeled with a 4×4 byte matrix, and a binary variable $\Delta X_i[j][k]$ is associated with every differential byte $\delta X_i[j][k]$. These binary variables are equal to 0 if their associated differential bytes are equal to 0^8, i.e.,

$$\Delta X_i[j][k] = 0 \Leftrightarrow X_i[j][k] = X'_i[j][k] \Leftrightarrow \delta X_i[j][k] = 0^8$$

and they are equal to 1 otherwise. We also associate binary variables $\Delta K_i[j][k]$ and $\Delta Y_i[j][k]$ with every differential byte $\delta K_i[j][k] = K_i[j][k] \oplus K'_i[j][k]$ and $\delta Y[j][k] = Y_i[j][k] \oplus Y'_i[j][k]$, respectively.

The operations that transform δX into δX_r (described in Sect. 2.1 and Fig. 1), are translated into constraints between these binary variables. In this first step, the goal is to find solutions which satisfy these constraints. Solutions of this first step are called *binary solutions*. Note that during this first step, the SubBytes operation S is not considered. Indeed, the S-box does not introduce nor remove differences, i.e., for all bytes A and B, $(A \oplus B = 0^8) \Leftrightarrow (S(A) \oplus S(B) = 0^8)$.

Second step: Search of byte solutions. In the second step, we try to transform binary solutions found in the first step into *byte solutions*. More precisely, for each binary variable $\Delta X_i[j][k]$, $\Delta Y_i[j][k]$ or $\Delta K_i[j][k]$ set to 1 in the binary solution, we search for a byte value $\delta X_i[j][k]$, $\delta Y_i[j][k]$ or $\delta K_i[j][k]$ different from 0^8 so that the AES transformation rules are satisfied. Note that some binary solutions may not be transformable into byte solutions. These binary solutions are called *byte-inconsistent binary solution*, whereas binary solutions that can be transformed into byte solutions are called *byte-consistent binary solutions*. Note also that a byte-consistent binary solution may be transformable into more than one byte solution.

Objective function. The goal is to find a byte solution that maximizes probability p_2 of Eq. (2), while being strictly lower than 1 (*i.e.*, there must be at least one difference between the initial plain texts and keys). It has been shown that a byte solution that maximizes probability p_2 also maximizes the number of factors $D_{\alpha,\beta}$ of Eq. (2) for which $\alpha = \beta = 0$ (because $D_{0^8,0^8} = 1$ whereas $D_{\alpha,\beta} \leq \frac{4}{256}$ if $(\alpha, \beta) \neq (0^8, 0^8)$). Therefore, we introduce a variable *obj* which is equal to the number of $\Delta X_i[j][k]$ and $\Delta K_i[3][k]$ variables of Eq. (2) which are set to 1:

$$obj = \sum_{i=0}^{r} \sum_{j=0}^{3} \sum_{k=0}^{3} \Delta X_i[j][k] + \sum_{i=0}^{r} \sum_{k=0}^{3} \Delta K_i[3][k]$$

We add the constraint $obj \geq 1$, to ensure that probability p_2 is strictly lower than 1. To find a byte solution that maximizes p_2, we first have to find byte-consistent binary solutions that minimize the value of *obj*. Note that there may exist byte-inconsistent binary solutions that have a smaller *obj* value. However, these binary solutions must be discarded as it is not possible to transform them into byte solutions. Finally, among all byte-consistent binary solutions that minimize *obj*, we have to search for the one that maximizes the actual probability p_2 associated with its best byte solution.

Existing approaches to solve the problem. In [3], step 1 is solved with a dedicated Branch & Bound approach. In a preliminary study, we have implemented this approach in C programming language. For $r = 3$ (resp. $r = 4$), we found the optimal binary solution in about one hour (resp. 24 h) on a single core PC. In [9], step 1 is solved by performing a breadth-first traversal of a state-transition graph that has about $2^{33.6}$ nodes for a 128 bit key length. The graph needs 60 GB of memory and it is pre-computed in 30 min on a 12-core computer for $r = 5$. Using this graph, binary solutions are found in a few seconds. Both in [3,9], it is claimed that the optimal solution for $r = 4$ rounds has an objective value $obj = 13$. We shall see in Sect. 6 that there exists a better solution.

3   First CP Model for Step 1

In this section, we describe a CP model for the first step described in Sect. 2.4. This model was initially introduced in [14].

3.1   Variables

Let r be the number of rounds, and let $l = 128$ be the length of the key. We define the following binary variables (see Fig. 1 for an overview of the bytes associated with these variables):

– For each column and row $j, k \in [0; 3]$, $\Delta X[j][k]$ is the variable associated with the differential byte $\delta X[j][k] = X[j][k] \oplus X'[j][k]$.

- For each round $i \in [0; r]$ and for each column and row $j, k \in [0; 3]$, $\Delta X_i[j][k]$ and $\Delta K_i[j][k]$ are variables associated with differential bytes $\delta X_i[j][k] = X_i[j][k] \oplus X_i'[j][k]$ and $\delta K_i[j][k] = K_i[j][k] \oplus K_i'[j][k]$, respectively.
- For each round $i \in [0; r - 1]$ and for each column and row $j, k \in [0; 3]$, $\Delta Y_i[j][k]$ is the variable associated with the differential byte $\delta Y_i[j][k] = Y_i[j][k] \oplus Y_i'[j][k]$.

All these variables are binary variables, which are set to 0 when the associated differential byte is 0^8 and to 1 otherwise.

3.2 Constraints

Constraints correspond to the propagation of differences by operations of the round function f. As said before, the non linear operation S does not imply any constraint as it neither introduces nor removes differences. The linear ARK and KS operations involve xor operations. Therefore, we first define a xor constraint. Then, we define constraints implied by the SR, MC, ARK and KS operations.

xor. Let us consider three differential bytes δA, δB and δC such that $\delta A \oplus \delta B = \delta C$. If $\delta A = \delta B = 0^8$, then $\delta C = 0^8$. If $(\delta A = 0^8$ and $\delta B \neq 0^8)$ or $(\delta A \neq 0^8$ and $\delta B = 0^8)$ then $\delta C \neq 0^8$. However, if $\delta A \neq 0^8$ and $\delta B \neq 0^8$, then we cannot know if δC is equal to 0^8 or not. When abstracting differential bytes δA, δB and δC with binary variables ΔA, ΔB and ΔC (which only model the fact that there is a difference or not), we obtain the following definition of the xor constraint:

$$xor(\Delta A, \Delta B, \Delta C) \Leftrightarrow \Delta A + \Delta B + \Delta C \neq 1$$

AddRoundKey. At the beginning of the ciphering process, ARK performs xor operations on ΔX and ΔK_0 to obtain ΔX_0, *i.e.*,

$$\forall (j, k) \in [0; 3]^2, xor(\Delta X[j][k], \Delta K_0[j][k], \Delta X_0[j][k])$$

Then, for each round $i \in [1, r]$, ARK performs xor operations on ΔY_{i-1} and ΔK_i to obtain ΔX_i, *i.e.*,

$$\forall i \in [1, r], \forall (j, k) \in [0; 3]^2, xor(\Delta Y_{i-1}[j][k], \Delta K_i[j][k], \Delta X_i[j][k])$$

ShiftRows and MixColumns. For each round $i \in [0, r-1]$, SR and MC are applied on ΔX_i to obtain ΔY_i, and MC ensures MDS (see Sect. 2.1), *i.e.*,

$$\forall i \in [0, r-1], \forall j \in [0; 3], \sum_{k=0}^{3} \Delta X_i[(k+j)\%4][k] + \Delta Y_i[j][k] \in \{0, 5, 6, 7, 8\}$$

KeySchedule. KS is applied at each round i to compute ΔK_i from ΔK_{i-1}, and it is composed of xor operations between some columns of the key, *i.e.*,

$$\forall i \in [1, r], \forall k \in [0; 3], xor(\Delta K_{i-1}[0][k], \Delta K_{i-1}[3][(k+1)\%4], \Delta K_i[0][k])$$
$$\forall i \in [1, r], \forall j \in [1; 3], \forall k \in [0; 3], xor(\Delta K_i[j-1][k], \Delta K_{i-1}[j][k], \Delta K_i[j][k])$$

3.3 Objective Variable

Finally, we introduce an integer variable obj, whose domain is $[1, \frac{l}{6}]^1$, and we define obj as the number of differential variables on which a non linear S operation is performed, $i.e.$,

$$obj = \sum_{i=0}^{r} \sum_{j=0}^{3} \sum_{k=0}^{3} \Delta X_i[j][k] + \sum_{i=0}^{r} \sum_{k=0}^{3} \Delta K_i[3][k]$$

3.4 Ordering Heuristics

As we want to minimize the number of $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ variables set to 1, we add a variable ordering heuristic that first assigns these variables, and a value ordering heuristic that first tries to assign them to 0.

3.5 Limitations of the First CP Model for Step 1

In [14], we evaluated the CP model described in Sect. 3 (implemented with Choco 3 [19]) on two problems: the optimization problem, the goal of which is to find a binary solution that minimizes the value of obj, and the enumeration problem, the goal of which is to find all binary solutions for a given value of obj (corresponding to the optimal one). These very first experimental results showed us that Choco is able to solve these problems up to $r = 5$ rounds in a reasonable amount of time. Note that it has been shown in [3] that it is useless to try to solve these problems for more than 5 rounds because no valid characteristics exist beyond this limit. However, solutions for low values of r are used as a basis to build attacks with larger values of r. For example, [3] shows how to build an attack for $r = 12$ and $l = 192$ by combining 2 solutions with $r = 4$. Hence, it is very useful to find solutions with lower values of r.

For these two problems, binary solutions are not necessarily byte-consistent. In particular, it may happen that the binary solution of the optimization problem is byte-inconsistent. For instance, for $r = 3$ rounds, the optimal binary solution has a cost of $obj = 3$ and there exist 512 binary solutions with this cost. However, none of these solutions are byte-consistent: The optimal byte-consistent binary solution has a cost of $obj = 5$. When solving the enumeration problem with this cost, we find 21,504 solutions, among which only 2 are byte-consistent. This means that we spend most of the time at generating useless binary solutions which are discarded in the second step because they are byte-inconsistent. Note that approaches proposed by [3,9] also suffer from the same problem.

---

[1] The upper bound $\frac{l}{6}$ comes from the fact that $D_{\alpha,\beta} \leq 2^{-6}, \forall (\alpha, \beta) \neq (0^8, 0^8)$, and probability p_2 must be larger than 2^{-l} which corresponds to a probability with uniform distribution of the 2^l possible keys.

4   Additional Constraints for Step 1

In this section, we introduce new variables and constraints that are added to the first CP model described in Sect. 3. They are used to infer equality relations between differential bytes, and these relations are used to propagate the MDS property of MixColumns at the byte level. They remove most binary solutions that cannot be transformed into byte solutions, thus speeding up the solution process.

4.1   Propagation of MDS at the Byte Level

For each round $i \in [0, r-1]$ and each column $j \in [0, 3]$, the MDS property of MixColumns (introduced in Sect. 2.1) ensures:

$$\sum_{k=0}^{3} w(X_i[(k+j)\%4][k]) + w(Y_i[j][k]) \in \{0, 5, 6, 7, 8\}$$

At differential byte level, this property still holds:

$$\sum_{k=0}^{3} w(\delta X_i[(k+j)\%4][k]) + w(\delta Y_i[j][k]) \in \{0, 5, 6, 7, 8\}$$

In the first model, this property is ensured by the constraint:

$$\sum_{k=0}^{3} \Delta X_i[(k+j)\%4][k] + \Delta Y_i[j][k] \in \{0, 5, 6, 7, 8\}$$

However, the MDS property also holds for any xor difference between two different columns in two different rounds of the differential byte model: $\forall i_1, i_2 \in [0, r-1], \forall j_1, j_2 \in [0, 3]$,

$$\sum_{k=0}^{3} w(\delta X_{i_1}[(k+j_1)\%4][k] \oplus \delta X_{i_2}[(k+j_2)\%4][k])$$
$$+ \ w(\delta Y_{i_1}[j_1][k] \oplus \delta Y_{i_2}[j_2][k]) \in \{0, 5, 6, 7, 8\}$$

To ensure this property (that removes most byte-inconsistent boolean solutions), we introduce new boolean variables, called equality variables: For each pair of differential bytes δA and δB (in δX_i, δY_i, and δK_i matrices), we introduce the boolean equality variable $EQ_{\delta A, \delta B}$ which is equal to 1 if $\delta A = \delta B$, and to 0 otherwise. Using these differential byte equality variables, the MDS property between different columns is ensured by the following constraint: $\forall j_1, j_2 \in [0, 3], \forall i_1, i_2 \in [0, r-1]$,

$$\sum_{k=0}^{3} EQ_{\delta X_{i_1}[(k+j_1)\%4][k], \delta X_{i_2}[(k+j_2)\%4][k]} + EQ_{\delta Y_{i_1}[j_1][k], \delta Y_{i_2}[j_2][k]} \in \{0, 1, 2, 3, 8\}$$

4.2   Constraints on Equality Variables

In this section, we define constraints that hold on equality variables.

Constraints derived from xor constraints. As pointed out in Sect. 3.2 when defining the constraint $xor(\Delta A, \Delta B, \Delta C)$ (where ΔA, ΔB and ΔC are binary variables associated with differential bytes δA, δB and δC, respectively), if $\Delta A = \Delta B = 1$, then we cannot know if ΔC is equal to 0 or 1. However, whenever $\Delta C = 0$ (resp. $\Delta C = 1$), we know for sure that the corresponding byte δC is equal to 0^8 (resp. different from 0^8), meaning that the two bytes δA and δB are equal (resp. different), *i.e.*, that $EQ_{\delta A, \delta B} = 1$ (resp. $EQ_{\delta A, \delta B} = 0$). The same reasoning may be done for ΔA and ΔB because $(\delta A \oplus \delta B = \delta C) \Leftrightarrow (\delta B \oplus \delta C = \delta A) \Leftrightarrow (\delta A \oplus \delta C = \delta B)$. Therefore, we redefine the xor constraint as follows:

$$
\begin{aligned}
xor(\Delta A, \Delta B, \Delta C) \Leftrightarrow &((\Delta A + \Delta B + \Delta C \neq 1) \\
\wedge\ &(EQ_{\delta A, \delta B} = 1 - \Delta C) \\
\wedge\ &(EQ_{\delta A, \delta C} = 1 - \Delta B) \\
\wedge\ &(EQ_{\delta B, \delta C} = 1 - \Delta A))
\end{aligned}
$$

Constraints to ensure that equality variables define an equivalence relation. Symmetry is ensured by

$$
\forall \delta A, \delta B, EQ_{\delta A, \delta B} = EQ_{\delta B, \delta A}
$$

and transitivity by

$$
\forall \delta A, \delta B, \delta C, (EQ_{\delta A, \delta B} = EQ_{\delta B, \delta C} = 1) \Rightarrow (EQ_{\delta A, \delta C} = 1)
$$

Constraints that relate equality variables with binary differential variables. For each pair of differential bytes $\delta A, \delta B$ such that the corresponding binary variables are ΔA and ΔB, respectively, we have:

$$
(EQ_{\delta A, \delta B} = 1) \Rightarrow (\Delta A = \Delta B)
$$
$$
EQ_{\delta A, \delta B} + \Delta A + \Delta B \neq 0
$$

4.3   Constraints Derived from KS

The KeySchedule (described in Sect. 2.1) mainly performs xor operations: At each round i, the first column $K_i[0]$ is obtained by performing a xor between bytes of $K_{i-1}[0]$ and $K_{i-1}[3]$; for the last three columns $j \in \{1, 2, 3\}$, $K_i[j]$ is obtained by performing a xor between $K_{i-1}[j]$ and $K_i[j-1]$. Besides these xor operations, all bytes of $K_{i-1}[3]$ pass through the S-box before xoring them with $K_{i-1}[0]$ to obtain $K_i[0]$. Therefore, each byte of K_i, for each round $i \in [1, r]$ may be expressed as a combination of xor operations between bytes of the initial

key K_0, and bytes obtained by applying the S operation on column 3 of rounds $j < i$. For example (recall that $A \oplus A = 0^8$ and $0^8 \oplus A = A$):

$$\begin{aligned}
K_2[1][1] &= K_2[0][1] \oplus K_1[1][1] \\
&= K_1[0][1] \oplus S(K_1[3][2]) \oplus K_1[0][1] \oplus K_0[1][1] \\
&= S(K_1[3][2]) \oplus K_0[1][1]
\end{aligned}$$

When reasoning at the differential byte levels, we have

$$\delta K_2[1][1] = \delta SK_1[3][2] \oplus \delta K_0[1][1]$$

where $\delta SK_1[3][2] = S(K_1[3][2]) \oplus S(K_1'[3][2])$. As S is a non linear operation, we cannot assume that $\delta SK_1[3][2] = S(\delta K_1[3][2])$. Therefore, $\delta SK_1[3][2]$ is a new differential byte. However, there is a finite number of such new differential bytes: for each round $i \in [0, r]$ and each line $k \in [0, 3]$, we introduce a new differential byte

$$\delta SK_i[3][k] = S(K_i[3][k]) \oplus S(K_i'[3][k])$$

and a new binary variable $\Delta SK_i[3][k]$ which is equal to 0 if $\delta SK_i[3][k] = 0^8$, and to 1 otherwise. Note that $\Delta SK_i[3][k]$ is a redundant variable which is equal to $\Delta K_i[3][k]$. So, we add the constraint

$$\forall i \in [1, r], \forall k \in [0, 3], \Delta K_i[3][k] = \Delta SK_i[3][k]$$

We introduce this redundant variable because at the byte level this equality no longer holds, *i.e.*, $\delta K_i[3][k] = A \nRightarrow \delta SK_i[3][k] = S(A)$ (because S is a non linear operator such that $S(A \oplus B) \neq S(A) \oplus S(B)$ except when $A = B$), and for the V sets defined below we reason at the byte level.

We propose to exploit the fact that each differential byte of K_i is the result of a xor between a finite set of bytes. We first use the KS rules defined in Sect. 2.1 to build, for each $i \in [1, r]$, and $j, k \in [0, 3]$, the set $V(i, j, k)$ of all differential bytes (coming either from δK_0 or from the set of new differential bytes δSK_i), such that:

$$\delta K_i[j][k] = \bigoplus_{\delta A \in V(i,j,k)} \delta A$$

For example, $V(2, 1, 1) = \{\delta K_0[1][1], \delta SK_1[3][2]\}$.

Note that these sets are computed before the search and do not depend on the initial values of plaintexts and keys.

For each of these sets, we introduce a set variable which contains the corresponding binary differential variables which are equal to 1:

$$V_1(i, j, k) = \{\Delta A \mid \delta A \in V(i, j, k) \wedge \Delta A = 1\}$$

For example, if $\Delta K_0[1][1] = 1$ and $\Delta SK_1[3][2] = 0$, then $V_1(2, 1, 1) = \{\Delta K_0[1][1]\}$.

Whenever two differential key bytes $\delta K_{i1}[j1][k1]$ and $\delta K_{i2}[j2][k2]$ have the same V_1 sets, then we may infer that $\delta K_{i1}[j1][k1] = \delta K_{i2}[j2][k2]$. More precisely, we define the constraint: $\forall i_1, i_2 \in [1, r], \forall j_1, j_2, k_1, k_2 \in [0, 3]$,

$$(V_1(i_1, j_1, k_1) = V_1(i_2, j_2, k_2)) \Rightarrow (EQ_{\delta K_{i_1}[j_1][k_1], \delta K_{i_2}[j_2][k_2]} = 1)$$

Also, if $V_1(i, j, k)$ is empty (resp. contains one or two elements), we infer that $\Delta K_i[j][k]$ is equal to 0 (resp. a variable, or a xor between 2 variables). More precisely, we define the constraints: $\forall i \in [1, r], \forall j, k \in [0, 3]$,

$$V_1(i, j, k) = \emptyset \Rightarrow \Delta K_i[j][k] = 0$$
$$V_1(i, j, k) = \{\Delta A\} \Rightarrow \Delta K_i[j][k] = 1 \wedge EQ_{\delta K_i[j][k], \delta A} = 1$$
$$V_1(i, j, k) = \{\Delta A, \Delta B\} \Rightarrow xor(\Delta A, \Delta B, \Delta K_i[j][k])$$

From a practical point of view, V_1 variables are not modeled with set variables, but with vectors of boolean variables. The dimension of these vectors is equal to the number of possible elements in these sets, $i.e.$, $16 + 4(r + 1)$ (the 16 bytes of K_0 plus the four bytes that pass through an S-box at each round). Each boolean variable $V[p]$ is equal to 1 if the pth element belongs to V_1 ($i.e.$, if the variable associated with the pth element is equal to 1), and to 0 otherwise. For each of these vectors, we introduce an integer variable which is constrained to be equal to the sum of the variables of the vector.

5   CP Model for Step 2

We have implemented in Choco 3 [19] the second step that, given a binary solution, searches for the byte-consistent solution with the highest p_2 value (or prove that there is no byte-consistent solution). The CP model for this second step is rather straightforward and mainly uses table constraints to define relations between the input and the output of the S-box function. The key point is to use a variable ordering that first chooses variables associated with the matrix ΔX_i such that $\sum_{j,k} \Delta X_i[j][k]$ is minimal.

The second step is not a bottleneck and is rather quickly solved by Choco. For example, when $l = 128$, it is solved in 0.41 (resp. 0.42 and 1.26) seconds, on average, when the number of rounds is $r = 3$ (resp. $r = 4$ and $r = 5$), whereas it is solved in 2.26 seconds when $l = 192$ and $r = 8$. Therefore, we have not tried to use other solvers for this step.

6   Experimental Evaluation

In this section, we experimentally compare our two CP models for Step 1: Model 1 refers to the first model introduced in Sect. 3; Model 2 refers to the first model plus the additional constraints introduced in Sect. 4. These two models are defined with the MiniZinc modelling language [18]. Model 2 is available at http://gerault.net/resources/CP_AES.tar.gz.

We compare three solvers on these models: Gecode [10] and Choco 4 [19], which are classical CP solvers, and Chuffed [4], which is a lazy clause hybrid solver that combines features of finite domain propagation and Boolean satisfiability. All solvers are run on a single core and with default parameters[2], except option `-f` for Choco 4 (to break ties of the heuristic described in Sect. 3.4 with the last conflict heuristic). All runs are limited to one hour of CPU time on a 2.5–3.5 GHz i7-4710MQ processor with 8 GB of memory.

Table 1 sums up the results for a number of rounds $r \in \{3, 4, 5\}$[3]. For each round, the objective value obj ranges from the largest value such that Model 1 finds no solution to the smallest value such that there exists a byte-consistent binary solution. Table 1 shows us that Model 2 drastically reduces the number of byte-inconsistent solutions: For example, there are more than $9 * 10^7$ byte-inconsistent solutions with Model 1 when $r=4$ and $obj=11$, whereas there is no solution with Model 2. Hence, Model 2 is much more efficient than Model 1.

For both models, the number of choice points is greater for Gecode than for Choco, and for Choco than for Chuffed. However, choices points are handled faster by Gecode than by Choco (probably because Choco is implemented in Java and Gecode in C++), and faster by Choco than by Chuffed (probably due to lazy clause generation overhead). Therefore, Choco is not faster than Gecode on small instances, and Chuffed is not faster than Choco on small or medium-size instances. For the hardest instance ($r=5$; $obj=17$), Chuffed is nearly twice as fast as Choco, which is nearly twice as fast as Gecode.

All solvers are much faster than the Branch & Bound approach of [3]: Our C implementation of this approach needs 24 h to find an optimal binary solution when $r = 4$. They are also faster and much less memory consuming than the approach of [9], that needs 60 GB and 30 min on a 12-core computer to pre-compute the graph. For example, for $r = 5$ and $obj = 17$, Choco and Chuffed need 400MB and 88MB, respectively.

New results for differential cryptanalysis. We have found two byte-consistent binary solutions with $obj = 12$ for $r = 4$ rounds, and we have proven the optimality of these solutions by showing that there does not exist another byte-consistent binary solution with an obj value strictly lower than 12. The optimal byte solution (computed in Step 2) when $obj = 12$ has a probability $p_2 = 2^{-79}$. The optimal byte solution and its associated binary solution are given in Appendix A. This solution is better than the solution claimed to be optimal in [3] and [9]: In these papers, authors say that the best byte-consistent binary solution for $r = 4$ has an obj value equal to 13, and that the optimal byte solution has a probability $p_2 = 2^{-81}$.

---

[2] We tried other parameter settings. The best results were obtained with default ones.

[3] Let us recall that it has been shown in [3] that it is useless to try to solve the problem for more than 5 rounds when the key length is $l = 128$.

Table 1. Comparison of models and solvers, on the enumeration problem. Each line displays: The number of rounds r, the objective function value obj, the number of byte-consistent binary solutions (S), and the results with models 1 and 2 (number of binary solutions (bin), CPU time in seconds (Time) and number of choice points (CP) for Gecode, Choco 4 and Chuffed). We report '-' when Time is greater than 3600.

r	obj	S	Model 1							Model 2						
			bin	Gecode		Choco 4		Chuffed		bin	Gecode		Choco 4		Chuffed	
				Time	CP	Time	CP	Time	CP		Time	CP	Time	CP	Time	CP
3	2	0	0	**0.0**	9E1	**0.0**	4E1	**0.0**	5E1	0	**0.0**	9E1	0.1	4E1	**0.0**	5E1
3	3	0	5E2	0.1	2E3	0.4	2E3	**0.0**	7E2	0	**0.0**	3E2	0.3	2E2	0.1	2E2
3	4	0	5E3	1.3	2E4	1.8	1E4	**0.2**	5E3	0	**0.2**	9E2	0.5	4E2	**0.2**	4E2
3	5	2	2E4	6.0	6E4	5.1	5E4	**0.9**	2E4	4	**0.4**	2E3	0.6	1E3	0.6	1E3
4	8	0	0	**0.2**	2E4	0.6	1E4	0.3	8E3	0	4.6	1E4	4.9	5E3	6.2	4E3
4	9	0	2E4	7.1	1E5	5.4	7E4	**1.4**	4E4	0	8.1	2E4	**7.8**	8E3	10.7	7E3
4	10	0	6E6	-	-	1161.2	2E7	**113.5**	6E6	0	14.2	3E4	**12.8**	1E4	16.2	1E4
4	11	0	9E7	-	-	-	-	**1974.5**	9E7	0	24.4	5E4	**15.5**	2E4	25.2	2E4
4	12	2	-	-	-	-	-	-	-	8	44.7	1E5	**28.4**	5E4	35.7	3E4
5	10	0	0	**1.1**	1E5	1.4	5E4	2.3	4E4	0	39.2	3E4	**26.8**	2E4	37.3	1E4
5	11	0	3E1	**2.0**	2E5	2.4	1E5	5.0	7E4	0	63.0	5E4	**46.4**	3E4	61.5	2E4
5	12	0	5E5	998.0	4E6	98.3	2E6	**48.4**	7E5	0	110.0	9E4	**74.6**	5E4	97.9	3E4
5	13	0	4E7	-	-	-	-	**1246.5**	5E7	0	187.4	2E5	**142.1**	9E4	157.6	5E4
5	14	0	-	-	-	-	-	-	-	0	321.7	3E5	247.4	2E5	**246.5**	8E4
5	15	0	-	-	-	-	-	-	-	10	586.7	5E5	448.2	3E5	**408.1**	1E5
5	16	0	-	-	-	-	-	-	-	35	1175.8	1E6	770.1	6E5	**593.5**	2E5
5	17	6	-	-	-	-	-	-	-	50	2879.0	5E6	1524.9	1E6	**885.1**	4E5

7 Discussion and Conclusion

We have introduced a CP model for solving a problem related to the chosen key differential cryptanalysis of AES with keys of length $l = 128$. This model follows the classical two step solving process of [3, 9]. In Step 1, we abstract bytes with binary values that indicate whether the byte is equal to 0^8. In Step 2, we search for non null byte values, for each binary value equal to 1. We have defined new constraints (not used in [3, 9]) which allow us to dramatically reduce the number of binary solutions that cannot be transformed into byte solutions. The idea is to keep track of equalities at the byte level to remove byte-inconsistent solutions at the binary level.

In this paper, we have described models for AES-128, with key length l=128. We have also defined MiniZinc models for AES-192, with l=192. At this time, the best solution we obtained for AES-192 concerns 8 rounds and has $obj = 19$ active S-boxes. We also plan to extend this work to other families of block ciphers, such as Rijndael [5] for which the approach of [9] cannot be used because of its exponential memory complexity.

In our model, we use boolean variables to represent equivalence classes defined by byte equalities: For each pair of bytes, we introduce a boolean variable which is set to 1 if the two bytes are equal, and we explicitly add constraints to ensure symmetry and transitivity of the equality relation. Another possibility

would have been to use a graph variable (whose nodes are differential bytes, and edges are byte equality relations), and to post an n-clique global constraint on it, as proposed by Fages [7]: This constraint ensures that the graph is composed of n disjoint cliques, where each clique corresponds to an equivalence class. We have not used this constraint in our model, as it is not available in MiniZinc. We plan to investigate the interest of this constraint using Choco.

Finally, the CP model for Step 2 mainly uses table constraints. Some AES operations operate at the bit level (mostly xor operations), and we plan to improve our model by using bit-vector variables and channeling them with integer variables used to model bytes, as proposed in [13].

Appendix

A Solution with $obj = 12$ Active S-Boxes for AES with $r = 4$ Rounds and $l = 128$ Bits

The byte-consistent binary solution is displayed below. Each binary variable assigned to 1 is colored in blue, and is surrounded in red when it belongs to the objective function (*i.e.*, it passes through an S-box).
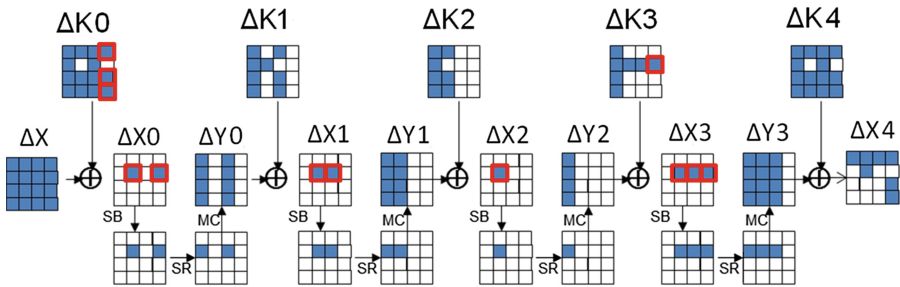
The optimal byte solution is displayed below, in hexadecimal notation.

Round	$\delta X = X \oplus X'$	$\delta K = K \oplus K'$
Init	0d151846 0dacf2f2 0dfff2f2 0dacf2f2	
0	00000000 00ac0000 00000000 00ac0000	0d151846 0d00f2f2 0dfff2f2 0d00f2f2
1	00000000 00ff0000 00ff0000 00000000	0dfff2f2 00ff0000 0dfff2f2 00000000
2	00000000 00ff0000 00000000 00000000	0dfff2f2 0d00f2f2 00000000 00000000
3	00000000 00ff0000 00ff0000 00ff0000	0dfff2f2 00ff0000 00ff0000 00ff0000
End/4	fa000000 faff0000 fa000000 f700f2f2	f7fff2f2 f700f2f2 f7fff2f2 f700f2f2

The corresponding plaintexts X and X' and keys K and K' are displayed below, in hexadecimal notation. The probability p_2 associated with these plaintexts and keys is $p_2 = 2^{-79}$ whereas it is equal to $p_2 = 2^{-81}$ in the solution given in [3,9] (solution with $obj = 13$ active S-boxes).

Round	K	K'
0	00000000 00000000 00000000 00000000	0d151846 0d00f2f2 0dfff2f2 0d00f2f2
1	62636363 62636363 62636363 62636363	6f9c9191 629C6363 6f639191 62636363
2	9b9898c9 f9fbfbaa 9b9898c9 f9fbfbaa	96676a3b f4fb0958 9b9898c9 f9fbfbaa
3	90973450 696ccffa f2f45733 0b0fac99	9d68c6a2 6993cffa f2b5733 0bf0ac99
4	ee60da7b 876a1581 759e42b2 7e91ee2b	19f92889 706ae773 8261b040 89911cd9

Round	X	X'
Init.	6b291f8d a800d3d7 f239d5a4 510035ef	663c07cb a5ac2125 ffc62756 5cacc71d
0	6b291f8d a800d3d7 f239d5a4 510035ef	6b291f8d a8acd3d7 f239d5a4 51ac35ef
1	e5000327 00796300 0079005c 0000005a	e5000327 00866300 0086005c 0000005a
2	2e2de80b 5186a759 e0d3cbb2 2b02c803	2e2de80b 5179a759 e0d3cbb2 2b02c803
3	5a74f2ae b979ce4a e286aa6a ea86647b	5a74f2ae b986ce4a e279aa6a ea79647b
End	c501f2fa 4095b6af cdd8f67b 4fadf0a4	3f01f2fa ba6ab6af 37d8f67b b8ad0256

References

1. Biham, E.: New types of cryptanalytic attacks using related keys. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 398–409. Springer, Heidelberg (1994)
2. Biham, E., Shamir, A.: Differential cryptanalysis of feal and N-Hash. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 1–16. Springer, Heidelberg (1991)
3. Biryukov, A., Nikolić, I.: Automatic search for related-key differential characteristics in byte-oriented block ciphers: application to AES, Camellia, Khazad and others. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 322–344. Springer, Heidelberg (2010)
4. Chu, G., Stuckey, P.J.: Chuffed solver description (2014). http://www.minizinc.org/challenge2014/description_chuffed.txt
5. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer, Heidelberg (2002)
6. De, D., Kumarasubramanian, A., Venkatesan, R.: Inversion attacks on secure hash functions using SAT solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 377–382. Springer, Heidelberg (2007)
7. Fages, J.-G.: On the use of graphs within constraint-programming. Constraints **20**(4), 498–499 (2015)
8. FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197. U.S. Department of Commerce/N.I.S.T (2001)
9. Fouque, P.-A., Jean, J., Peyrin, T.: Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 183–203. Springer, Heidelberg (2013)

10. Team, G.: Gecode: Generic constraint development environment (2006). http://www.gecode.org
11. Karpman, P., Peyrin, T., Stevens, M.: Practical free-start collision attacks on 76-step SHA-1. IACR Cryptology ePrint Archive 2015:530 (2015)
12. Legendre, F., Dequen, G., Krajecki, M.: Encoding hash functions as a sat problem. In: IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, 7–9 November 2012, pp. 916–921. IEEE (2012)
13. Michel, L.D., Van Hentenryck, P.: Constraint satisfaction over bit-vectors. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 527–543. Springer, Heidelberg (2012)
14. Minier, M., Solnon, C., Reboul, J.: Solving a Symmetric Key Cryptographic Problem with Constraint Programming. In: ModRef 2014, Workshop of the CP 2014 Conference, September 2014, Lyon, France, July 2014
15. Mironov, I., Zhang, L.: Applications of SAT solvers to cryptanalysis of hash functions. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 102–115. Springer, Heidelberg (2006)
16. Morawiecki, P., Srebrny, M.: A sat-based preimage analysis of reduced keccak hash functions. Inf. Process. Lett. **113**(10–11), 392–397 (2013)
17. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C.-K., Yung, M., Lin, D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer, Heidelberg (2012)
18. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.R.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)
19. Prudhomme, C., Fages, J.-G.: An introduction to choco 3.0: an open source java constraint programming library. In: CP Workshop on CP Solvers: Modeling, Applications, Integration, and Standardization (2013)
20. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 244–257. Springer, Heidelberg (2009)
21. Sun, S., Hu, L., Wang, M., Yang, Q., Qiao, K., Ma, X., Song, L., Shan, J.: Extending the applicability of the mixed-integer programming technique in automatic differential cryptanalysis. In: López, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 141–157. Springer, Heidelberg (2015)
22. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014)
23. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
24. Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 1–16. Springer, Heidelberg (2005)

Solving a Supply-Delivery Scheduling Problem with Constraint Programming

Katherine Giles and Willem-Jan van Hoeve[✉]

Tepper School of Business, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
kgiles@tepper.cmu.edu, vanhoeve@andrew.cmu.edu

Abstract. We describe a constraint programming approach for a supply-delivery problem in the petrochemical industry, in which barges transport liquid material from supplier locations to downstream processing plants. The problem is to design a pickup-and-delivery route for each barge such that given minimum and maximum inventory levels at each location are met for a given fleet size. This optimization problem is part of a larger planning system to determine the fleet size, negotiate pickup windows and quantities, and design operational schedules. We evaluate our model on representative supply networks provided by BP North America, and contrast our results with those obtained by a mixed-integer programming approach.

1 Introduction

In the chemical processing industry, often material has to be processed by multiple plants in order to convert it into the final product. We consider such supply-delivery problem in the petrochemical industry, in which liquid material needs to be transported from supplier locations to downstream processing plants. The liquid material is transported by water, using so-called "tows" that consist of one power unit and two barges. We focus on the design of pickup-and-delivery schedules for the tows with the aim of satisfying minimum and maximum inventory levels at each of the supply locations and processing plants. The high-level objective is to minimize total cost, which is determined by the fleet size, the violation of time window constraints, and the violation of inventory (lower and upper) capacity constraints.

As a specific application, we study the supply-delivery problem that is operated by BP (formerly known as British Petroleum) in North America. BP employs the optimization problem described above within a larger planning system to determine the target fleet size, and to negotiate pickup windows and quantities. In addition, the solution to the optimization problem provides a basis for the operational schedule. Upon the start of the project, BP employed a mixed-integer programming (MIP) optimization model which had several drawbacks. First, the MIP model had difficulty finding optimal (or even good feasible) solutions, already for relatively small instances. Second, larger instances posed challenges with respect to memory issues. Third, given the large computation

times, it was difficult to use the MIP model in the desired strategic planning context. In order to find solutions of sufficient quality, this approach required intensive manual interaction: The model would be solved in several iterations, in each of which the problem parameters (such as the fleet size or pickup windows) would be adjusted.

Given these challenges, and driven by the scheduling aspects of the problem, we therefore present an alternative optimization model based on constraint programming (CP). In particular, we will represent the problem as a constraint-based scheduling problem, using activities and resources. Our main finding is that the CP model scales much better than the MIP formulation that is currently in place. For certain representative instances considered, CP can find optimal solutions in a fraction of the time it takes MIP. The CP model is therefore much better suited than the existing MIP model to provide solutions in a broader planning context.

The remainder of the paper is structured as follows: In Sect. 2 we provide a brief review of the most relevant literature. We then give a detailed description of our problem in Sect. 3. This is followed by the constraint programming model in Sect. 4. We provide an evaluation of our model in Sect. 5, and conclude in Sect. 6.

2   Related Work

Our problem can be viewed as a variant of the maritime inventory routing problem. The basic maritime inventory routing problem (see [3]) involves the transportation of a single product from loading ports to unloading ports, with each port having a given inventory storage capacity and a production or consumption rate, therefore combining inventory management and ship routing. These are typically treated separately in much of the maritime transportation industry. Christiansen and Fagerholt [4] provide a recent survey in ship routing and scheduling research.

A recent paper by Goel et al. [5] introduces a constraint programming model for a maritime inventory routing problem in the context of liquefied natural gas (LNG) tanker scheduling. It follows the approach of representing the routing problem as a constraint-based scheduling model, which has been successfully applied before to many industrial routing and scheduling problems [1,2]. In particular, the routing problem is represented as a disjunctive scheduling problem, in which a visit to a location becomes a task to be scheduled, and the distance between two locations is modeled as a 'sequence-dependent set-up time' between tasks. In addition, the same tasks can be associated with resource constraints to model the inventory levels over time. The effectiveness of CP scheduling models for such complex inventory routing problems was one of the main motivations for considering CP for our problem.

Our problem does differ considerably from that in [5], and from the standard maritime inventory routing problem. First, we consider multiple products instead of a single commodity. Furthermore, the attributes of the three location types

(suppliers, plants, and customers) are not simply pickup and delivery nodes; the plants both consume and produce products, necessitating both pickup and delivery visits. Lastly, our vessels have two barges, each of which can carry a separate product.

3   Problem Description

We will next describe the specific application at BP in detail [6]. The supply network is defined by a set of suppliers S, a set of plants P, a set of customer locations C, and a set of products F; see Fig. 1 for a schematic representation of a typical instance with four suppliers, two plants, and two customer locations. The suppliers provide 'feed' to the plants (to be picked up in specified time windows), which then convert this into different final products. In the example of Fig. 1, the set of products is $F = \{$feed, product 1, product 2, product 3$\}$. For convenience, we define $F^- := F \setminus \{$feed$\}$ to be the set of final products only. The final products are shipped from plants to customer locations to meet the demand (or to avoid inventory overflow at the plant). In addition, some plants can directly meet the demand of a product instead of shipping it to a customer location. For example, in Fig. 1, Plant 2 only produces final product 1, whose demand is met exclusively by prescheduled pickups. Lastly, some plants are connected to a pipeline which provides an alternate supply of feed (for example, Plant 1 in Fig. 1). Both pipeline deliveries and prescheduled pickups are external events; no tows will be used for these activities.

We are given a discrete planning horizon (in days) $H = \{1, 2, \ldots, \hat{H}\}$, where $\hat{H}$ represents the end of the horizon. We are also given a set of tows $T = \{1, 2, \ldots, M\}$, where M is the maximum number of tows. Each tow consists of two barges and a power unit, and has a total capacity of 2Λ metric ton. The specific attributes (input data) for the suppliers, plants, customers, and tows are given in Table 1. The goal is to design a pickup-and-delivery schedule for the tows such that (ideally) all time window constraints and all inventory capacity constraints are respected.

While the structure provided by this network is very generic, our case has the following specific elements:[1]

– In our model, we do not explicitly convert feed into final products. Instead, the conversion is modeled implicitly by specifying daily production and consumption amounts. (Note that other materials are required for the conversion as well.)
– Suppliers are contractually bound to have sufficient feed available for any scheduled tow pickup that can occur over the scheduling horizon. We therefore do not need to represent the daily production of feed, nor the inventory levels, at each supplier location. (Our model is easily extended to handle this, however.)

---

[1] Via private communication with Norman Jerome, BP Americas.
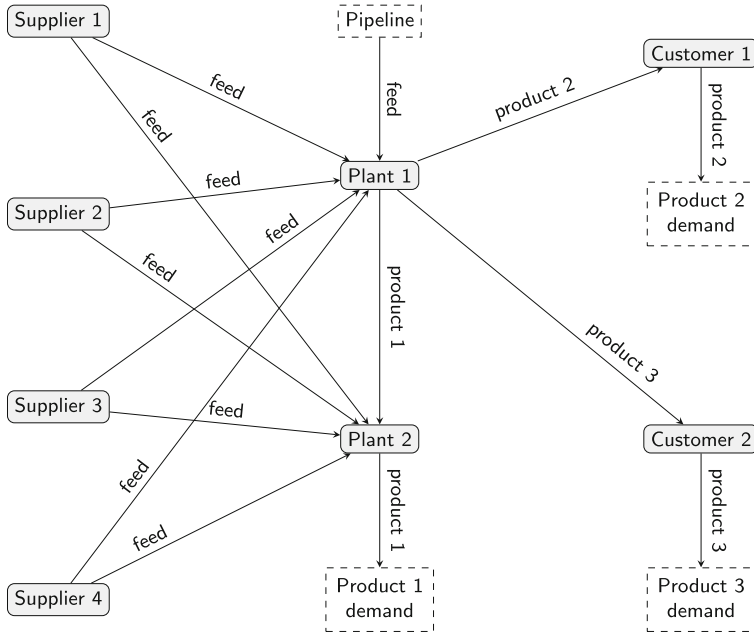
Fig. 1. Petrochemical supply-delivery network.

Table 1. Attributes of suppliers, plants, customers, and tows.

W_s	Set of pickup windows for supplier $s \in S$,
$l_{i,s}$	Start date of pickup window $i \in W_s$ for $s \in S$,
$u_{i,s}$	End date of pickup window $i \in W_s$ for $s \in S$,
$V_{p,f}^{\mathrm{pu}}$	Set of prescheduled visits to pick up product $f \in F^-$ for plant $p \in P$,
V_p^{del}	Set of prescheduled visits to deliver feed for plant $p \in P$,
$d_{p,f,i}^{pu}$	Date of visit $i \in V_p^{\mathrm{pu}}$ for $p \in P$ and product $f \in F^-$,
$d_{p,i}^{del}$	Date of visit $i \in V_p^{\mathrm{del}}$ for $p \in P$,
$a_{p,f,i}^{\mathrm{pu}}$	Amount of $f \in F^-$ picked up at visit $i \in V_{p,f}^{\mathrm{pu}}$ for $p \in P$,
$a_{p,i}^{\mathrm{del}}$	Amount of feed delivered at visit $i \in V_p^{\mathrm{del}}$ for $p \in P$,
$I_{f,i}^{\mathrm{init}}$	Initial inventory of product $f \in F$ at location $i \in P \cup C$,
$I_{f,i}^{\mathrm{min}}$	Minimum inventory of product $f \in F$ at location $i \in P \cup C$,
$I_{f,i}^{\mathrm{max}}$	Maximum inventory of product $f \in F$ at location $i \in P \cup C$,
$\Pi_{f,p,d}$	Production amount of product $f \in F$ at plant $p \in P$ on date $d \in H$,
$\Gamma_{p,d}$	Consumption amount of feed at plant $p \in P$ on date $d \in H$,
$D_{i,j}$	Distance (in days) from location i to location j $(i,j \in S \cup P \cup C)$,
l_t	First destination for tow $t \in T$,
Λ	Barge capacity (500 metric ton in our data case)

- The most important operational concern is to have sufficient feed available at the plants (i.e., the lower limit is typically binding). The upper capacity of the feed inventory at the plants is typically not binding.
- The maximum inventory capacity of the final products at the plants is typically not reached.
- The consumption rate of the final products at the customer locations is not part of our problem description. At the same time, the maximum inventory level is practically never binding for customer locations.

Given these considerations, as our specific objective we chose to minimize the total amount of feed underflow at the plant locations. However, we do present a generic CP model that can accommodate different objectives as well.

4   Constraint Programming Model

We next present our constraint programming model for the base problem: Given a fixed number of tows, find a supply-delivery schedule that minimizes the total feed underflow at the plants. We use the optimization modeling system AIMMS to express our model, using 'activities' and 'resources' for the constraint-based scheduling formulation [2]. We will first review the relevant AIMMS syntax, and then provide the details of the model.

4.1   AIMMS Syntax for Constraint-Based Scheduling

The activities and resources in AIMMS provide an interface to advanced scheduling constraints, in particular those available in IBM ILOG CP Optimizer [7]; readers familiar with the IBM ILOG CP Optimizer scheduling interface will recognize the similarities. Activities correspond to tasks to be executed over the time horizon. They have a start time, end time, and duration. Each activity also has a *schedule domain* which defines the range of possible dates for the start and end time. In addition, an activity can be *optional*, which means that its presence will be a decision variable. Activities can impact one or more resources; this is modeled at the resource level. In AIMMS, an activity A defines the following decision variables:

A.`Begin`   the start time of A,
A.`End`   the end time of A,
A.`Length`   the duration of A,
A.`Present`   the presence of A (with Boolean domain).

Several useful functions on activities are provided. For activity A and $d \in H$ we have:

- `ActivityBegin(`A`, `d`)`: Returns d if A is absent, and A.Begin if A is present.

Resources can be declared in two ways in AIMMS: Sequential or parallel. A sequential resource maintains a unary resource level (either 0 or 1), which means that at most one task can be active at a time. The definition of a sequential resource includes the following attributes:

– Resource name and index set (possibly empty),
– *Activities:* List of activities that influence the resource,
– *Group set:* Set of groups in which the activities are divided,
– *Group definition:* Maps activities to group set elements,
– *Group transition:* Represents the transition/setup time between pairs of group elements,
– *First activity:* Reference to the first activity in the sequence.

Sequential resources have several useful associated functions. For sequential resource R, activity A, and $l, d \in H$, we have:

– `BeginOfNext(R,A,l,d)`: Returns d if A is absent, l if A is present and scheduled as last activity on R, and B.`Begin` if A is present and not scheduled as last activity on R, and B is the next activity of A scheduled on R.
– `EndOfNext(R,A,l,d)`: Returns d if A is absent, l if A is present and scheduled as last activity on R, and B.`End` if A is present and not scheduled as last activity on R, and B is the next activity of A scheduled on R.

For sequential resource R, activity A, and group elements l, l', we have:

– `GroupOfNext(R,A,l,l')`: Returns l' if A is absent, l if A is present and scheduled as last activity on R, and the group of B on R if A is present and not scheduled as last activity on R, and B is the next activity of A scheduled on R.

For parallel resources, multiple tasks can be active simultaneously, as long as the activity level of the resource is within a given lower and upper bound. The activity level represents the cumulative resource value over time, and is influenced by the activities. The definition of a parallel resource includes the following attributes:

– Resource name and index set (possibly empty),
– *Activities:* List of activities that influence the resource,
– *Level range:* $\{L..U\}$ specifies that the activity level must be between L and U at each time point,
– *Initial level:* Specifies the initial activity level,
– *Begin change:* Specifies for each activity A by how much the activity level is changed at A.`Begin`,
– *End change:* Specifies for each activity A by how much the activity level is changed at A.`End`.

A useful function for a resource R and a time point $d \in H$ is the following:

– $R(d)$.`ActivityLevel`: Returns the activity level of R at time d.

This function uses 'overloading' of the resource name; for example, if we define a resource indexed over i as $R(i)$, we can access the activity level at time $d \in H$ via $R(i, d)$.`ActivityLevel`.

We remark that the description above is limited to those concepts that are relevant to our paper. For a complete description of the scheduling functionality in AIMMS we refer to [8].

4.2   Modeling the Location Visits and Inventory Levels

There are three different types of locations: Suppliers, who are solely pickup nodes, customers, who are exclusively delivery nodes, and plants, which both produce and consume product and, as such, are both pickup and delivery nodes.

Location Visits. Each possible visit by a tow to a location is indexed by a master set $N := \{1, 2, \dots\}$, for which the upper value is the maximum number of pickup windows for all the suppliers. That is, each tow cannot make more than $|N|$ visits to a location. One of the drivers of the computational efficiency of our model, however, is defining an auxiliary set $N_l \subseteq N$, with the maximum number of individual tow visits varying by location $l \in S \cup P \cup C$. For a supplier $s \in S$, the maximum number is equal to the number of pickup windows $|W_s|$, whereas for plants and customers, the maximum number can be adjusted by the user through the graphical interface, if desired. Restricting the set of location visits also restricts many of the index domains, so keeping its cardinality low greatly improves computational performance.

Integer Variables. Due to the multiple products and split load functionality, we require additional variables to represent the amount of material picked up and delivered at each visit. We let $L_f \subseteq P \cup C$ be the set of locations that serve as demand point for final product $f \in F^-$. The list of variables we will use to represent the pickup and delivery at the locations is given in Table 2.

Table 2. Variables for Modeling the Pickup and Delivery at the Locations.

$b^1_{t,i,f,p}$	amount of $f \in F^-$ picked up by barge 1 for $t \in T$ at plant p for visit $i \in N_p$,
$b^2_{t,i,f,p}$	amount of $f \in F^-$ picked up by barge 2 for $t \in T$ at plant p for visit $i \in N_p$,
$b^{\mathrm{pu}}_{t,i,f,p}$	total amount of $f \in F^-$ picked up for $t \in T$ at plant p for visit $i \in N_p$,
$b^{\mathrm{dem}}_{t,l,i,f}$	total amount of $f \in F^-$ delivered for $t \in T$, location $l \in L_f$, and $i \in N_l$,
$c_{p,d}$	amount of feed consumed at plant $p \in P$ on date $d \in H$,
$s_{p,d}$	amount of feed shortage (underflow) at plant $p \in P$ on date $d \in H$,
$g^{\mathrm{del}}_{p,i}$	amount of feed delivered at plant $p \in P$ for visit $i \in V^{\mathrm{del}}_p$,
$g_{t,p,i}$	amount of feed delivered with $t \in T$ at plant $p \in P$ for visit $i \in N_p$

As we must distinguish between barge capacity and tow capacity, we introduce the following constraints, for $t \in T, p \in P, i \in N_p, f \in F^-$:

$$0 \leq b^1_{t,i,f,p} \leq \Lambda,$$
$$0 \leq b^2_{t,i,f,p} \leq \Lambda,$$
$$b^{\mathrm{pu}}_{t,i,f,p} = b^1_{t,i,f,p} + b^2_{t,i,f,p}.$$

In addition, we ensure that each barge can only contain one product:

$$\sum_{f \in F^-} (b^1_{t,i,f,p} > 0) \leq 1 \qquad \text{for } t \in T, p \in P, i \in N_p,$$

$$\sum_{f \in F^-} (b^2_{t,i,f,p} > 0) \leq 1 \qquad \text{for } t \in T, p \in P, i \in N_p.$$

To model the feed shortage, we define the next constraints, for $p \in P, d \in H$:

$$0 \leq c_{p,d} \leq \Gamma_{p,d}, \tag{1}$$

$$0 \leq s_{p,d} \leq \Gamma_{p,d}, \tag{2}$$

$$c_{p,d} + s_{p,d} = \Gamma_{p,d}. \tag{3}$$

Unlike products, each tow picks up a "full load" (two barges) upon a supplier visit. Therefore, to model the delivery of feed with tows, we introduce the constraint $0 \leq g_{t,p,i} \leq 2\Lambda$, for $t \in T, p \in P, i \in N_p$. The other constraints for this purpose will be given in Sect. 4.5, as they depend on the definition of plant visit activities.

Activities. The key activities in our model are the possible visits that each tow can make to each location. We define the following optional activities representing the i-th visit of tow t to the respective locations:

$$\text{VisitSupplier}(t, s, i) \text{ for } t \in T, \ s \in S, \ i \in N_s,$$
$$\text{VisitPlant}(t, p, i) \quad \text{for } t \in T, \ p \in P, \ i \in N_p,$$
$$\text{VisitCust}(t, c, i) \quad \text{for } t \in T, \ c \in C, \ i \in N_c.$$

Each of these activities has a fixed duration of one day, and a uniquely defined schedule domain. For $\text{VisitSupplier}(t, s, i)$, the schedule domain includes only those days that are part of the pickup windows. $\text{VisitPlant}(t, p, i)$ and $\text{VisitCust}(t, c, i)$ have schedule domains that begin either the day the first prescheduled tow is due to arrive, or the earliest day a tow can arrive with delivery of feed or product, derived from the travel time from the closest supplier/plant production node.

We also define (fixed) activities to represent the daily production or consumption, as well as the prescheduled visits, at each location:

$$\text{PlantProduction}(p, f, d) \text{ for } p \in P, \ f \in F^-, \ d \in H,$$
$$\text{PlantConsumption}(p, d) \quad \text{for } p \in P, \ d \in H,$$
$$\text{PreSchedPickUp}(p, f, i) \quad \text{for } p \in P, \ f \in F^-, \ i \in V^{\text{pu}}_{p,f},$$
$$\text{PreSchedDelivery}(p, i) \quad \text{for } p \in P, \ i \in V^{\text{del}}_p.$$

Each of these activities has a fixed duration of one day, and must be present. The activities $\text{PlantProduction}(p, f, d)$ and $\text{PlantConsumption}(p, d)$ must take place on day d, i.e., their associated start time variable is fixed to d. The start time variables of activities $\text{PreSchedPickUp}(p, f, i)$ and $\text{PreSchedDelivery}(p, i)$ are fixed to $d^{\text{pu}}_{p,f,i}$ and $d^{\text{del}}_{p,i}$, respectively. Therefore, these activities can be viewed

as constants; they are necessary to model the resource levels, but their associated variables are fixed to given values.

We note that the activity $\texttt{PlantConsumption}(p, f, d)$ reduces the inventory level by $c_{p,d}$; any consumption that would reduce the inventory level below the lower bound is considered $s_{p,d}$, or underflow, as defined by constraints (1)–(3). A similar overflow variable could be modeled for $\texttt{PlantProduction}(p, f, d)$, but in this application, we found it more efficient to set hard upper bounds on product inventory levels and force the plants to ship product to customers to relieve any excess.

Location Resources. We incorporate the above production, consumption, tow visits, and prescheduled events in defining resources to represent the inventory levels of the various products at each location, as follows:

Parallel resource: $\texttt{PlantProductInventory}(p, f)$ for $p \in P, f \in F^-$
     Activities: $\texttt{PlantProduction}(p, f, d)$ for $d \in H$, $\texttt{VisitPlant}(t, p, i)$
              $\texttt{PreSchedPickup}(p, f, i)$ for $t \in T$, $i \in N_p$
    Level range: $\{I_{f,p}^{\min} \,..\, I_{f,p}^{\max}\}$
    Initial level: $I_{f,p}^{\text{init}}$
    End change: $\texttt{PlantProduction}(p, f, d)$: $\Pi_{f,p,d}$
              $\texttt{VisitPlant}(t, p, i)$: $-b_{t,i,f,p}^{\text{pu}}$
              $\texttt{PreSchedPickup}(p, f, i)$: $-a_{p,f,i}^{\text{pu}}$

Parallel resource: $\texttt{PlantFeedInventory}(p)$ for $p \in P$
     Activities: $\texttt{PlantConsumption}(p, d)$ for $d \in H$, $\texttt{VisitPlant}(t, p, i)$,
              $\texttt{PreSchedDelivery}(p, i)$, for $t \in T$, $i \in N_p$
    Level range: $\{I_{\text{feed},p}^{\min} \,..\, I_{\text{feed},p}^{\max}\}$
    Initial level: $I_{\text{feed},p}^{\text{init}}$
    End change: $\texttt{PlantConsumption}(p, d)$: $-c_{p,d}$
              $\texttt{VisitPlant}(t, p, i)$: $g_{t,p,i}$
              $\texttt{PreSchedDelivery}(p, i)$: $g_{p,i}^{\text{del}}$

Parallel resource: $\texttt{CustomerProductInventory}(c, f)$
     Activities: $\texttt{VisitCust}(t, c, i)$, for $t \in T$, $i \in N_c$
    Level range: $\{0 \,..\, I_{f,p}^{\max}\}$
    Initial level: $I_{f,c}^{\text{init}}$
    End change: $\texttt{VisitCust}(t, c, i)$: $b_{t,c,i,f}^{\text{dem}}$

To ensure that at most one tow can visit each plant or supplier at a time, we introduce the following sequential resources:

Sequential resource: $\texttt{PlantVisits}(p)$ for $p \in P$
       Activities: $\texttt{VisitPlant}(t, p, i)$, for $t \in T$, $i \in N_p$

Sequential resource: $\texttt{SupplierVisits}(s)$ for $s \in S$
       Activities: $\texttt{VisitSupplier}(t, s, i)$, for $t \in T$, $i \in N_s$

The supplier visits are special in that the set of possible visits N_s is defined by the set of pickup windows W_s for supplier $s \in S$, which is not the case for

plants. Therefore, we distribute these possible visits over the tows by adding the following constraints to ensure that at most one tow can visit a supplier in each pickup time window:

$$\sum_{t \in T} \texttt{VisitSupplier}(t, s, i).\texttt{Present} \le 1 \qquad \text{for } s \in S, i \in N_s. \qquad (4)$$

We recall that the specific pickup time windows are represented by the schedule domain of $\texttt{VisitSupplier}(t, s, i)$, which is defined as $\{l_{i,s}, \ldots, u_{i,s}\}$ for $t \in T$, $s \in S$, $i \in N_s$.

4.3   Modeling the Tows

For each tow, we define a unary resource for the sequence of visits (the route) and parallel resources for the inventory levels of each product carried by the tow. We assume that during each supplier pickup visit, both barges are filled with feed to full capacity (this was given as a requirement).

   We first define additional activities $\texttt{TowFirstAct}(t)$ for $t \in T$, which represent the starting location for each tow. As the scheduling process is dynamic, this activity accounts for tows *en route* at the beginning of the planning horizon. These prescheduled tows, even if not active for the entire schedule horizon, are the minimum number of total tows that can be hired. Any additional tows are not prescheduled, and begin at a depot with a one-day travel time to any location.

   We can now define the sequential resource representing the tow's route as follows:

Sequential resource: $\texttt{RouteSeq}(t)$ for $t \in T$
         Activities: $\texttt{VisitSupplier}(t, s, i)$, $\texttt{VisitPlant}(t, p, i)$
                  $\texttt{VisitCust}(t, c, i)$, $\texttt{TowFirstAct}(t)$
        Group set: $S \cup P \cup C$
  Group definition: $\texttt{VisitSupplier}(t, s, i)$: s,
                  $\texttt{VisitPlant}(t, p, i)$: p,
                  $\texttt{VisitCust}(t, c, i)$: c,
                  $\texttt{TowFirstAct}(t)$: l_t
  Group transition: $(\texttt{i},\texttt{j})$: $D_{i,j}$ for $i, j \in S \cup P \cup C$
      First activity: $\texttt{TowFirstAct}(t)$

   As tows carry several types of inventory, including both feed and final products, each of which is modeled as a different resource, it is necessary to use variables rather than parameters upon any plant visit. Hence, while the tow feed inventory level increases by a constant amount upon visiting a supplier, on any plant visit activity, a tow may or may not be delivering feed. Consequently, to model the tow inventory for feed, we define:

Parallel resource: $\texttt{TowFeedInv}(t)$
       Activities: $\texttt{VisitPlant}(t, p, i)$, $\texttt{VisitSupplier}(t, s, i)$
     Level range: $\{0 .. 2\Lambda\}$
  Begin change: $\texttt{VisitPlant}(t, p, i)$: $-g_{t,p,i}$
    End change: $\texttt{VisitSupplier}(t, s, i)$: 2Λ

We similarly model the tow inventory for each final product as follows:

Parallel resource: $\texttt{TowProdInv}(t, f)$ for $t \in T, f \in F^-$
      Activities: $\texttt{VisitPlant}(t, p, i), \texttt{VisitCust}(t, c, i)$
    Level range: $\{0 \mathinner{\ldotp\ldotp} 2\Lambda\}$
   End change: $\texttt{VisitPlant}(t, p, i)$: $b_{t,i,f,p}^{\text{pu}} - b_{t,p,i,f}^{\text{dem}}$
               $\texttt{VisitCust}(t, c, i)$: $-b_{t,c,i,f}^{\text{dem}}$

4.4 Additional Sequencing Constraints

There are two broad categories of additional sequencing constraints, both of which prune the search tree by limiting the options $\texttt{RouteSeq}(t)$ has after processing each activity. Neither category contains constraints that are strictly required to model the problem; however, their inclusion improves solver performance.

Visit Sequencing. We introduce the following constraints for the plant visits, for $t \in T, p \in P, i \in N_p \setminus \{1\}$:

$$\texttt{VisitPlant}(t, p, i).\texttt{Present} \Rightarrow \texttt{VisitPlant}(t, p, i-1).\texttt{Present} \quad (5)$$

$$\texttt{ActivityBegin}(\texttt{VisitPlant}(t, p, i), \hat{H}) > \quad (6)$$
$$\texttt{EndOfNext}(\texttt{RouteSeq}(t), \texttt{VisitPlant}(t, p, i-1), \hat{H}-1, \hat{H}-1)$$

Constraint (5) is a symmetry-breaking constraint that ensures we schedule the visits in order of N_p. Constraint (6) ensures we cannot schedule two consecutive visits of tow t at plant p, which prevents idling. We define similar constraints for the customer visits, for all $t \in T, c \in C, i \in N_c \setminus \{1\}$:

$$\texttt{VisitCust}(t, c, i).\texttt{Present} \Rightarrow \texttt{VisitCust}(t, c, i-1).\texttt{Present} \quad (7)$$

$$\texttt{ActivityBegin}(\texttt{VisitCust}(t, c, i), \hat{H}) > \quad (8)$$
$$\texttt{EndOfNext}(\texttt{RouteSeq}(t), \texttt{VisitCust}(t, c, i-1), \hat{H}-1, \hat{H}-1)$$

Location Sequencing. In addition to defining the order of visits, our model also restricts the locations a tow can visit after a given activity. As visits to suppliers leave a tow with no available capacity, we introduce the following constraints that make sure we visit a plant after a supplier, for $t \in T, s \in S, i \in N_s$:

$$(\texttt{BeginOfNext}(\texttt{RouteSeq}(t), \texttt{VisitSupplier}(t, s, i), \hat{H}, \hat{H}) \neq \hat{H}) \Rightarrow \quad (9)$$
$$\texttt{GroupOfNext}(\texttt{RouteSeq}(t), \texttt{VisitSupplier}(t, s, i)) \in P$$

Likewise, we introduce a constraint to model that we visit a demand location after picking up some product at a plant. We then define, for $t \in T, p \in P, i \in N_p$:

$$(\texttt{BeginOfNext}(\texttt{RouteSeq}(t), \texttt{VisitPlant}(t, p, i), \hat{H}, \hat{H}) \neq \hat{H} \quad (10)$$
$$\wedge (b_{t,i,f,p}^{\text{pu}} > 0) \Rightarrow (\texttt{GroupOfNext}(\texttt{RouteSeq}(t), \texttt{VisitPlant}(t, p, i)) \in L_f \setminus \{p\})$$

It is also possible to deliver final products at multiple locations in sequence. Since we must first empty both barges before we can pick up new feed from a supplier (a given requirement), we introduce the following constraints for $t \in T$, $p \in P, i \in N_p$:

$$((\texttt{BeginOfNext}(\texttt{RouteSeq}(t), \texttt{VisitPlant}(t,p,i), \hat{H}, \hat{H}) \neq \hat{H}) \;\wedge \qquad (11)$$

$$\bigvee_{f \in F^-} (\texttt{TowProdInv}(t, f, \texttt{VisitPlant}(t,p,i).\texttt{Begin}).\texttt{ActivityLevel} > 0)) \Rightarrow$$

$$\texttt{GroupOfNext}(\texttt{RouteSeq}(t), \texttt{VisitPlant}(t,p,i)) \in \cup_f L_f,$$

and for $t \in T, c \in C, i \in N_c$:

$$((\texttt{BeginOfNext}(\texttt{RouteSeq}(t), \texttt{VisitCust}(t,c,i), \hat{H}, \hat{H}) \neq \hat{H}) \;\wedge \qquad (12)$$

$$\bigvee_{f \in F^-} (\texttt{TowProdInv}(t, f, \texttt{VisitCust}(t,c,i).\texttt{Begin}).\texttt{ActivityLevel} > 0)) \Rightarrow$$

$$\texttt{GroupOfNext}(\texttt{RouteSeq}(t), \texttt{VisitCust}(t,c,i)) \in \cup_f L_f.$$

In practice, these constraints can be refined to operate on subsets of locations and products (and their associated activities). For example, in our case, customer sites only receive one type of final product. Note that the model does not direct a tow's route after both barges are empty, to allow the tows to "choose" to visit a supplier to pick up more feed, or a plant to pick up more final product.

4.5 Linking Pickup and Delivery Amounts with Visits

We introduce the following constraints to model the delivery amount of feed at the plants, for $t \in T, p \in P, i \in N_p$:

$$(\texttt{VisitPlant}(t,p,i).\texttt{Present} = 0) \Rightarrow (g_{t,p,i} = 0) \qquad (13)$$

$$(\texttt{VisitPlant}(t,p,i).\texttt{Present} = 0) \Rightarrow (\sum_{f \in F^-} b_{t,p,i,f}^{\text{dem}} = 0) \qquad (14)$$

$$(\texttt{VisitPlant}(t,p,i).\texttt{Present} = 0) \Rightarrow (\sum_{f \in F^-} b_{t,i,f,p}^{\text{pu}} = 0) \qquad (15)$$

$$(\texttt{VisitPlant}(t,p,i).\texttt{Present} = 1) \Rightarrow \qquad (16)$$
$$(\texttt{TowFeedInv}(t, \texttt{VisitPlant}(t,p,i).\texttt{End}).\texttt{ActivityLevel} = 0)$$

Constraints (13), (14) and (15) state that no feed or product can be delivered, or product can be picked up, when the plant is not visited. Constraint (16) states that partial deliveries are not allowed (this is a given requirement).

Similar constraints can be defined for the customer locations:

$$(\texttt{VisitCust}(t,c,i).\texttt{Present} = 0) \Rightarrow (\sum_{f \in F^-} b^{\text{dem}}_{t,c,i,f} = 0) \tag{17}$$

$$(\texttt{VisitCust}(t,c,i).\texttt{Present} = 1) \Rightarrow (\sum_{f \in F^-} b^{\text{dem}}_{t,c,i,f} > 0) \tag{18}$$

$$(\texttt{VisitCust}(t,c,i).\texttt{Present} = 1) \Rightarrow \tag{19}$$
$$(\texttt{TowProdInv}(t, \texttt{VisitCust}(t,c,i).\texttt{End}).\texttt{ActivityLevel} = 0)$$

Constraint (17) ensures that we cannot deliver any product when the customer is not visited. Constraint (18) on the other hand states that we must deliver a product when the customer is visited, and constraint (19) makes sure that no partial delivery occurs.

4.6   Objective

The objective function is deceptively simple. In our implementation of the model, we chose to minimize the total shortage of feed at the plants, as that was the most pressing issue apparent:

$$\min \sum_{p \in P, d \in H} s_{p,d}. \tag{20}$$

As we will see in our experimental evaluation, the specific instances we were given permit solutions in which there is no feed shortage, i.e., given a sufficient number of tows, our solutions satisfy all pickup window, underflow, and overflow constraints. However, solving the problem as a constraint optimization problem with objective (20) proved much more computationally efficient than solving the associated constraint satisfaction problem in which feed shortage is not allowed.

We do note, however, that the objective can easily be adapted to include other terms, depending on the application at hand.

5   Evaluation

We implemented our CP model in Aimms 4.20, using IBM ILOG CPLEX and CP Optimizer (12.6.3) as MIP and CP solver, respectively. We performed an evaluation on the supply network given in Fig. 1. We consider two representative cases over the same 92-day schedule horizon. Both cases have seven tows en route and 20 total tows available; additionally, both have similar production and demand profiles. They differ in the initial location and availability of the tows as well as in some of the dates of supplier pickup windows.

We compare our CP model with the MIP model that is currently in use at BP [6], as described in the introduction. The objective of the MIP model is to minimize total cost, which is a weighted sum of the number of tows in use, the violation of inventory upper and lower capacity constraints (overflow and

underflow), and violation of pickup time windows. The tows account for the largest relative cost in the objective.

The MIP model is given the option to use all 20 tows available, which, together with minimizing the constraint violations, allows to find some feasible solutions early in the solving process. The CP model, on the other hand, uses a fixed number of tows; in the reported experiments we use the minimum required number of tows (seven) as well as eight tows.

Table 3. Comparing CP and MIP solutions on two representative cases.

	CP				MIP	
	Case 1		Case 2		Case 1	Case 2
	7 tows	8 tows	7 tows	8 tows		
Variables	7,042	7,593	7,000	7,545	466,564	466,816
Constrains	3,387	3,836	3,371	3,822	509,068	509,320
Number of tows	7	8	7	8	8	7
Total underflow (mt)	0	0	0	0	200	17.72
Total overflow (mt)	0	0	0	0	0	11.83
Time window violations	0	0	0	0	0	5
Optimality gap					18.40 %	13.69 %
Solving time (s)	385	313	1,140	170	3,600	3,600

The results are presented in Table 3. The table first shows for each model the number of variables and constraints. In addition, in order to compare CP with MIP, we report for each solution the number of tows used, the total underflow and overflow, and the number of time window violations, i.e., visits to suppliers outside the defined pickup windows. The last two rows indicate the optimality gap (for MIP) and total solving time. Both MIP models were unable to solve the cases optimally within a time limit of 3,600 s, whereas all CP models were solved optimally, in some cases within a couple minutes. Furthermore, the solutions provided by CP satisfy all problem constraints, whereas the solutions found by MIP use more tows for Case 1 (8 tows instead of 7), and violate various constraints.

AIMMS not only offers an optimization modeling language, but also comes with visualization tools to build an end-user interface. The use of the model as a planning tool is facilitated by a graphical user input page which allows the user to vary the maximum number of tow and location visits, as discussed earlier, and, most importantly, the maximum number of tows. The input page accompanies several additional visualizations, including the inventory profiles at the locations, as well as the routing schedules for the tows. Figures 2 and 3 provide an illustration; they depict the inventory profiles for Plant 1 (feed) and Plant 2 (product 1) and the routing sequence for eight tows, in an optimal solution.
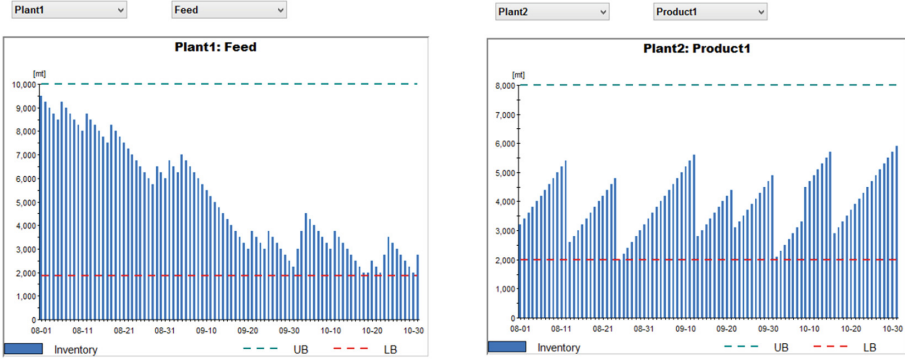
Fig. 2. Inventory levels for Plant 1's feed (left) and Plant 2's product 1 (right).
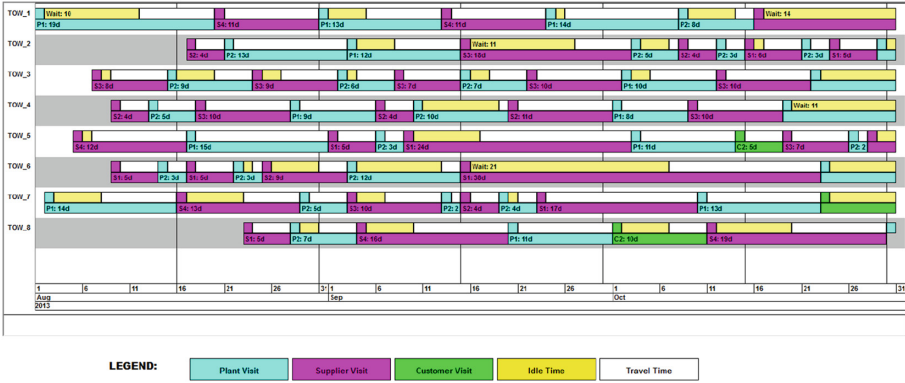
Fig. 3. Gantt chart representing the routes for the tows.

6   Conclusion

We introduced a constraint programming model for a maritime inventory routing problem in the petrochemical sector. Our model is based on a constraint-based scheduling formulation, and relies on activities to represent possible vessel visits, and parallel resources to represent the inventory levels over time. In addition, we utilize sequential resources to represent the vessel routes. As a case study we considered a barge scheduling problem from BP, with supplier, plant, and customer locations. We compared the performance of our CP model to an existing MIP formulation. The CP model was able to find optimal solutions with a small fleet size that satisfied all problem constraints, while the MIP formulation had scalability issues, and was not able to return solutions of similar quality even when given much more computation time.

References

1. Baptiste, P., Laborie, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling and planning. In: Rossi, F.. van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, chap. 22. Elsevier (2006)
2. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer Academic Publishers (2001)
3. Christiansen, M., Fagerholt, K.: Maritime inventory routing problems. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, pp. 1947–1955. Springer, New York (2009)
4. Christiansen, M., Fagerholt, K., Nygreen, B., Ronen, D.: Ship Routing and Scheduling in the New Millenium. Eur. J. Oper. Res. **228**, 467–483 (2013)
5. Goel, V., Slusky, M., van Hoeve, W.-J., Furman, K., Shao, Y.: Constraint programming for LNG ship scheduling and inventory management. Eur. J. Oper. Res. **241**(3), 662–673 (2015)
6. Jerome, N.: Description of a MIP Supply Delivery Scheduling Problem (2014). Unpublished manuscript
7. Laborie, P.: IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 148–162. Springer, Heidelberg (2009)
8. Roelofs, M., Bisschop, J.: AIMMS - The Language Reference. AIMMS (2016)

Four-Bar Linkage Synthesis Using Non-convex Optimization

Vincent Goulet[1]([✉]), Wei Li[2], Hyunmin Cheong[2], Francesco Iorio[2], and Claude-Guy Quimper[1]

[1] Université Laval, Quebec, Canada
`vincent.goulet.4@ulaval.ca`
[2] Autodesk Research, Toronto, Canada

Abstract. We show how four-bar linkages can be designed using non-convex optimization techniques. Our generative design software takes as input a curve that needs to be reproduced by a four-bar linkage and outputs the best assembly that approximates this curve. We model the problem using quadratic constraints and show how redundant constraints help to solve the problem. We also provide an algorithm that samples the curve based on its characteristics. Experiments show that our software is faster and more precise than existing systems. The current work is part of a larger generative design initiative at Autodesk Research.

1 Introduction

A mechanism is an arrangement of machine parts that generates a specified motion. The synthesis of a mechanism is the process of determining the position, the orientation, and other parametric properties of parts according to constraints governing their alignment or motion. The design of mechanisms has greatly benefited from the advent of computer techniques. Computer-aided design and engineering software (CAD/CAE) have been widely used in the documentation, analysis, and optimization of designs. Though, still nowadays, the existing technologies and tools lack a function for automated mechanism synthesis. Creating mechanisms that meet specified motion and geometric requirements demands highly trained expert designers. As the available computing power keeps growing, so does the interest in the development of generative design tools.

In this paper, we address the problem of generating a four-bar linkage that outputs a prescribed curve (Fig. 1). Four-bar linkages are simple yet practically important mechanisms that can generate complex motion. Since the First Industrial Revolution, they have been widely applied in mechanical systems, including manufacturing, agriculture, robotics, and automotive industry [16]. However, it is a laborious process to manually design a four-bar linkage based on a target curve. The current state of the art of four-bar linkage design is a time consuming process, and the results often lack optimality or generality. Hence, this paper introduces an automated and efficient four-bar linkage synthesis approach, which is also an important milestone towards synthesizing more complex mechanisms.
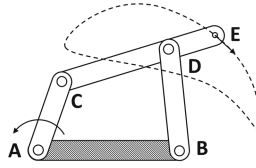
Fig. 1. A four-bar linkage and output *coupler curve*

We first establish the current state of the art regarding mechanical assembly and path synthesis. We then introduce the terminology and notation used for the four-bar linkage and the global non-convex optimization strategy. The contributions presented are the feature identification sampling technique, the four-bar linkage quadratic model, the special constraint on the area of the curve, and the design software developed. Results emphasizing the speed and quality of our method follow along with a discussion.

2   Related Work

2.1   Mechanical Assembly

A long standing challenge of mechanical design is the automation of design synthesis tasks [21]. Existing mechanism synthesis methods include systematic search [17,22], machine-learning based approaches [8], stochastic search [26] and graph-based approach [18,20]. However, to the best of our knowledge, existing generative design approaches still lack the generality or performance to be practical.

2.2   Path Synthesis of Four-Bar Linkage

Without a computer-aided approach, a human designer typically uses prior knowledge and/or atlases of coupler curves to identify a candidate linkage to produce the desired curve [16]. Then the chosen linkage is modified until it is satisfactory [25].

Analytical approaches formulate the four-bar linkage constraints, solve the problem and return exact solutions [19,24]. They require a set of points or positions input by the user, which can be challenging to provide. In many cases, there is no mechanism that can produce exactly the desired path. In fact, although the mechanism found goes through the specified points, it may not go through the desired curve (see Fig. 8 for an example). Also, analytical approaches are limited to solving problems with five or less target points, (see [14] and the references therein).

Alternatively, numerical methods are used to synthesize approximate mechanisms with acceptable tolerance between the input path and the coupler curve. Genetic algorithms have been widely applied to the four-bar mechanism synthesis problem [2,7]. Genetic algorithms and other stochastic search methods [6]

have the same limitation – there is no assurance that they will find a global optimum. Also, because the objective function of four-bar mechanisms is highly constrained, the typical evolutionary algorithms have to choose a very large number of initial population so that a considerable amount of them can play in the next iteration. This technique unnecessarily increases CPU time and reserves a large amount of memory during the computing iterations. The lack of consistency also makes it challenging for performance evaluation.

Machine-learning approaches [8, 25] store a large number of coupler curves in a database. Automated procedures for fitting coupler curves are used to locate potential linkage solutions from the database. Neural network [12] and sequential quadratic programming [8] can be used to match coupler curves. However, such an approach requires building a large linkage database. Another limitation is that the quality of the generated motion directly depends on the mechanisms in the database and sampling techniques.

3   Preliminaries

3.1   Mechanical Linkage

A mechanical linkage is a set of rigid bodies, called *links*, connected by joints. Though many types of joints exist, we herein only consider the *revolute* joint or *pivot*, which allows for one degree of freedom rotation. This paper focuses on the two-dimensional four-bar linkage (Fig. 1), made of four links in a closed loop. The joints **A**, **B**, **C** and **D** are pivots. The positions of **A** and **B** are fixed. The motion of point **E** is the output of the mechanism, therefore it is called the *end effector*. The link **AB**, called the *frame*, cannot move. The link **AC**, called the *crank*, drives the motion of the linkage. The link **BD** is driven back and forth about **B** and is called the *rocker*. The link **CDE**, called the *coupler*, couples the rocker to the crank. The path traced by **E** over a full rotation of the crank is called the *coupler curve*. A four-bar linkage is *collinear* if point **E** is aligned with **C** and **D**. A linkage is *Grashof* if one link is able to fully rotate. We assume this condition is met and the crank **AC** can fully rotate.

3.2   Non-convex Global Optimization

Mathematical optimization aims at finding good solutions to a problem according to some user-defined criteria. *Global* optimization is a family of techniques that guarantee that the solutions returned are absolutely optimal. These techniques often consist of relaxing the problem to a form efficiently solvable to optimality. This relaxed solution provides a bound for *branch and bound search*.

To be computable, the *global optimization problem* is modelled mathematically. The model consists of a set of variables, a set of constraints that need to be satisfied, and an objective function. Each variable has a *domain*, a set of all values it can be assigned.

The *solver* takes as input the model and finds suitable values for all variables, such that constraints are satisfied and the objective is optimized. Solvers are

available for a wide range of applications and can be categorized by the types of variables they can handle, whether Boolean, integer, or real. Solvers can also be categorized by the types of functions they can handle, whether logic, linear, convex, or non-convex.

Modelling a four-bar linkage requires real variables and non-convex constraints. The global optimization solver *Couenne* [5] is specialized in both regards, and is the solver used for all experimentation presented. It was chosen over related candidates of comparable performance such as Baron [23] and AlphaBB [4] because it is open source. The constraint solver IBEX [1] was also considered but did not show sufficient performance. Other considered continuous solvers include RealPaver [10], SCIP [3] and LindoAPI [15].

Non-convex problems are difficult to solve even for the best available software. Couenne combines many techniques from constraint programming and other optimization subfields. It uses constraint propagation and interval arithmetic to achieve bounds tightening on each variable [5], therefore reducing the search space. It relaxes the non-linear constraints into linear envelopes. It uses branch and bound to create more tightly bounded subproblems. By adding redundant constraints, this envelope is further tightened. Whether redundant constraints make the solving faster depends on their number and complexity. Testing is required for validation. Couenne feeds the linear problem to CPLEX [13] to compute the solution to the relaxation.

4   Contribution

We developed a strategy to effectively design four-bar linkages outputting a desired curve using non-convex optimization. The benefits of this application are that the synthesis of the continuous curve is accurate, fast, and deterministic. The time frame of this project spanned six months. The first two months were used to survey the available technology. The last four months were used to develop the model and strategy. We modelled the mechanism using its geometric properties, keeping in mind the possible generalization to mechanisms of higher complexity, and a novel cut (or redundant constraint) was developed using the area of the curve. We also designed a novel point sampling technique. We implemented this strategy in a simple design software.

4.1   Fitness Metric

We aim at designing a four-bar linkage which replicates as tightly as possible a continuous curve. To make this problem tractable for the constraint solver, we strategically sample the curve using the technique describe in Sect. 4.2. The model described in Sect. 4.3 minimizes a single variable e which represents the maximum distance from the curve to a sample point. We sample the curve with as little points as possible to keep the search space small. Note that the solver could return a solution with zero error, meaning the solution curve reaches all sample points, and yet not match the input curve, as shown in Fig. 8. In general,
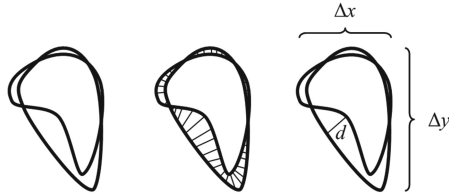
Fig. 2. The Hausdorff distance is obtained by finding, for all points on one curve, the closest point on the other, and keeping the distance of the furthest pair. X- and y-dimensions are also shown.

(a)   (b)   (c)

Fig. 3. $Q =$ (a) 1.7 %; (b) 4.4 %; (c) 20.8 %

it is necessary to evaluate how well the continuous input curve matches the output curve after it is returned, regardless of the objective value.

Several well-established curve matching metrics exist. The Hausdorff distance [11] d is the greatest distance from any point on the curves to the closest point on the other curve. To render the metric independent of the size of the curves, we normalize it with the greatest x- or y-dimension of the curve. The normalized Hausdorff distance is herein designated as Q. In compliance with Fig. 2, the equation for Q is:

$$Q = \frac{d}{\max\left(\varDelta x, \varDelta y\right)}$$

Figure 3 shows matching curves for different Q values. A Q value of 0 is a perfect match. The user can define a threshold T under which the curves are considered a good match. It is worth noting that Q does not consider the course of the curve, which might result in undesired matches, especially when a curve self-crosses. However, features of the model such as the area constraint discussed in Sect. 4.4 make these events unlikely.

4.2   Curve Sampling Technique

The input curve is pre-sampled into a high resolution array of coordinates. The sampling process consists of choosing n points from this array. Here we propose a strategy to choose the n sample points $T_1, \ldots, T_n$ that best represent the input curve. We call the number n of sample points the *sample number*. A compromise needs to be made when choosing the sample number. Indeed, large sample numbers increase the execution time. However, they also improve the Q value as they better represent the continuous curve.

Some points are more important than others, like cusps or sharp turns. We call these points of interest *features*. Figure 4 shows a curve with features and one without. It is also important to have some sample points between the features to depict the general behaviour. The remaining sampling points are spread evenly between the features. It is possible for a curve to have no feature (e.g. an ellipse). In this case, the samples are distributed uniformly with one placed at the point of maximal curvature (Fig. 4).
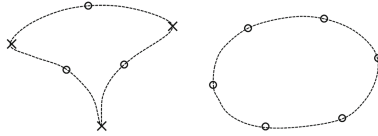
Fig. 4. Sampling technique: features are marked by × and remaining points by ○.

To identify the features, we find all maxima of curvature. However, we do not compute the actual curvature, because it approaches infinity at cusps and reaches inconveniently high values at very sharp turns. Instead, as shown on Fig. 5, we compute the squared change in angle θ^2 between segments of the high resolution pre-sampled array. We square θ to amplify the variation.
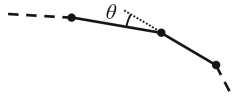
Fig. 5. The deviation θ between consecutive segments

We compute the median absolute deviation from all squared angles θ^2. Using the first derivative of θ^2 with respect to the distance travelled on the curve, we identify the local maxima. There are usually many extrema, and filtering is needed. We keep only the extrema whose θ value is significantly greater than the overall values over the curve. Experience has shown that filtering out data within 10 times the median absolute deviation yields satisfactory results. Algorithm 1 presents the equivalent pseudocode.

Algorithm 1. Feature filtering$(\boldsymbol{x}, \boldsymbol{y})$

1: $\Theta \leftarrow \{\theta_i^2 \mid \theta_i \text{ is the exterior angle at } (x_i, y_i)\}$
2: $\hat{\Theta} \leftarrow \{\theta_i^2 \in \Theta \mid \theta_{i-1}^2 < \theta_i^2 > \theta_{i+1}^2\}$
3: $m \leftarrow \text{median}(\Theta)$
4: $d \leftarrow \text{median}\{|\theta_i^2 - m| \mid \theta_i^2 \in \Theta\}$
5: **return** $\{(x_i, y_i) \mid \theta_i^2 \in \hat{\Theta} \wedge \theta_i^2 > m + 10d\}$

4.3   Model

The model ensures that the effector **E** moves as close as possible to the target curve. It minimizes the distance when the effector passes to each of the n sample points. In other words, the solver has to find a mechanism and compute n positions for this mechanism. Each position brings the effector close to its corresponding target point. This section describes the variables, constraints and objective function that compose the model.

Variables. A collinear four-bar linkage is defined by eight parameters, which are the x- and y-coordinates of pivots **A** and **B**, the lengths of links **AC**, **BD**, and **CD**, and the distance from **C** to **E**. The variable for the length between two points such as **AC** is denoted AC. The solved linkage is interpreted directly from the values of these variables. We nevertheless define two more redundant variables. The variable AB represents the distance between **A** and **B**. The variable w gives the ratio of length CE over CD.

  We add to the model variables for the x- and y-coordinates of **C**, **D**, and **E** for each target point, for a total of $6n$ variables. A single error variable e represents the maximum of all distances between effector positions $\mathbf{E}_i$ and their corresponding sample point $\mathbf{T}_i$.

  Without loss of generality, all variables for coordinates are bounded from -10 to 10. Link lengths are bounded from 0 to 10. The ratio w is bounded from 0 to 4. It is helpful for the algorithm's filtering to bound the domain of e with the upper error bound e_u.

  The information on the variables is gathered in Table 1.

Table 1. Variables for four-bar linkage model

Type	Variables	Domains	Quantity
Defining parameters	A_x, A_y, B_x, B_y	$[-10, 10]$	4
	$AB,\ AC,\ BD,\ CD,\ CE$	$[0, 10]$	5
	w	$[0, 5]$	1
Position parameters	$C_x, C_y, D_x, D_y, E_x, E_y$	$[-10, 10]$	$6n$
Error	e	$[0, e_\mathrm{u}]$	1

Constraints are relationships between the variables. The solver must find values for the variables to satisfy all constraints. Here we explain all the constraints of our model.

  The first set of constraints force the coordinates to be separated by distances corresponding to the lengths of the bars. For example, for the crank **AC** we have:

$$(A_x - C_{x_i})^2 + (A_y - C_{y_i})^2 = AC^2 \qquad\qquad \forall\, i \in [1, n]$$

We use a similar constraint to define the error e as the upper bound of the squared distance from points $\mathbf{E}_i$ to points $\mathbf{T}_i$.

$$(T_{x_i} - E_{x_i})^2 + (T_{y_i} - E_{y_i})^2 \leq e \qquad \forall\, i \in [1, n]$$

The following constraints ensure that the points $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{E}$ are collinear. We use the fact that the components of vectors $\mathbf{CE}$ and $\mathbf{CD}$ respect the ratio w.

$$w \cdot (D_{x_i} - C_{x_i}) = E_{x_i} - C_{x_i} \qquad \forall\, i \in [1, n]$$
$$w \cdot (D_{y_i} - C_{y_i}) = E_{y_i} - C_{y_i} \qquad \forall\, i \in [1, n]$$

The lengths of the bars are not sufficient to determine the configuration of the mechanism. As shown in Fig. 6, the same bars can be arranged into two distinct mechanisms. The two solutions share a symmetry along the segment joining $\mathbf{B}$ and $\mathbf{C}$. For each target point $\mathbf{T}_i$, the coordinates for $\mathbf{E}_i$ have to be on the same side of the segment $\mathbf{BC}_i$. Since $\mathbf{T}_i$ and $\mathbf{E}_i$ must lie close to one another, constraining either one is equivalent. The cross-product of vectors $\mathbf{BC}$ and $\mathbf{BE}$ changes sign depending on which side of $\mathbf{BC}$ the point $\mathbf{E}$ is. By constraining the sign of the cross-product to be the same for all positions, we constrain the configuration. We therefore add either of the next two constraints.

$$(T_{x_i} - C_{x_i})(B_y - C_{y_i}) \geq (T_{y_i} - C_{y_i})(B_x - C_{x_i}) \qquad \forall\, i \in [1, n] \qquad (1)$$
$$(T_{x_i} - C_{x_i})(B_y - C_{y_i}) \leq (T_{y_i} - C_{y_i})(B_x - C_{x_i}) \qquad \forall\, i \in [1, n] \qquad (2)$$

The two constraints are mutually exclusive, so a model may represent one configuration at a time. To access the whole search space, we can run the two configurations in parallel. We term them the *left* and *right* configurations, according to the inequality sign.
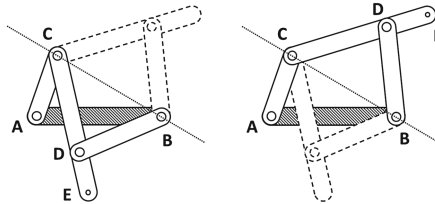
Fig. 6. The two possible configurations of the same links

The Grashof condition [9] states that the shortest link in a four-bar linkage can fully rotate only if the combined length of the shortest and longest links is smaller than the combined length of the remaining two links. The following constraints enforce this:

$$AB \geq AC \qquad\qquad BD \geq AC \qquad\qquad CD \geq AC$$
$$CD + BD \geq AC + AB + s$$
$$AB + CD \geq AC + BD + s$$
$$AB + BD \geq AC + CD + s$$

A security constant s is added to the three last constraints to avoid equality. Otherwise, the mechanism could fold over itself completely. In this state, its behaviour is indeterminate, as it can unfold in two ways, as shown in Fig. 7. It is the singularity where the mechanism can switch between the two configurations of Fig. 6. Singularities are generally undesirable as they require additional control and involve high mechanical stress. The security constant s can be tuned to the desired tolerance. For all experiments herein s was set to 0.1, to minimally reduce the search space while preventing singularities.
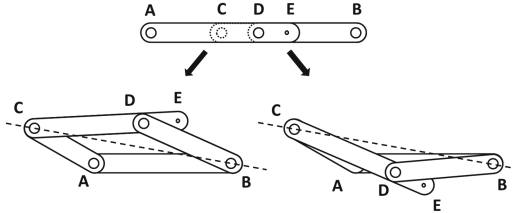
Fig. 7. Singular behaviour when $CD + BD = AC + AB$

It is worth noting that the model does not require the solution found to follow the sample points in any order. Therefore, in theory, the solver could return a mechanism which goes through the sample points in an undesired order.

Table 2. List of constraints for four-bar linkage model

Constraint	Quantity
$(A_x - B_x)^2 + (A_y - B_y)^2 = AB^2$	1
$(A_x - C_{x_i})^2 + (A_y - C_{y_i})^2 = AC^2$	n
$(B_x - D_{x_i})^2 + (B_y - D_{y_i})^2 = BD^2$	n
$(C_{x_i} - E_{x_i})^2 + (C_{y_i} - E_{y_i})^2 = CE^2$	n
$w \cdot (D_{x_i} - C_{x_i}) = E_{x_i} - C_{x_i}$	n
$w \cdot (D_{y_i} - C_{y_i}) = E_{y_i} - C_{y_i}$	n
$AB \geq AC$	1
$BD \geq AC$	1
$CD \geq AC$	1
$CD + BD \geq AC + AB + s$	1
$AB + CD \geq AC + BD + s$	1
$AB + BD \geq AC + CD + s$	1
$(T_{x_i} - C_{x_i}) \cdot (B_y - C_{y_i}) \lessgtr$ $(T_{y_i} - C_{y_i}) \cdot (B_x - C_{x_i})$	n
$(T_{x_i} - E_{x_i})^2 + (T_{y_i} - E_{y_i})^2 \leq e$	n

However, this is unlikely for two reasons. First, part of the sampling is done by filling gaps between the features. This creates a continuity between the points which the solutions tend to follow naturally. Second of all, violating the order of the points generally results in a significant change in the curve area, which is constrained as discussed in Sect. 4.4 (Table 2).

Objective Function. The goal is to minimize the distance between the two continuous curves, using the continuous metric Q. Implementing Q in the model would require approximating the curve with a very large number of points. To avoid enlarging the model, we have opted for using only carefully selected points to approximate the curve.

The variable e is defined in an analogous way to Q, but for a low resolution approximation of the curve. We therefore choose the objective function of minimizing e, which is the maximum distance between the output and input curves at the sample points. Therefore, even though ultimately we want to minimize the continuous metric Q, we had to define a discrete metric for the solver to use.

4.4 Constraint on Area

So far, the information contained about the curve is limited to the sample points. There is a chance that the solution found may go through the sample points, yet not produce the desired output (see Fig. 8). If we add more points for a tighter fit, the model will grow proportionately, with added variables and non-convex constraints. In general, it is desirable that the search space be as small as possible while sacrificing little precision.
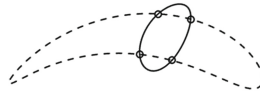
Fig. 8. Possible solution to a curve with few sample points

A relationship found empirically allowed including the area of the curve in the model. Figure 9(a) shows that the area of the coupler curve varies linearly with the ratio w of CE over CD. Thus, an expression of the following form can be induced:

$$Area(w) = a \cdot w + b$$

To determine a and b, two points are needed. First, when w is 0, the end effector **E** coincides with **C** and the coupler curve is a circle with radius AC. Second, when w is 1, the **E** coincides with point **D** and moves on an arc of null area (Fig. 9(b)).

$$Area(0) = \pi \cdot AC^2 \qquad\qquad Area(1) = 0$$
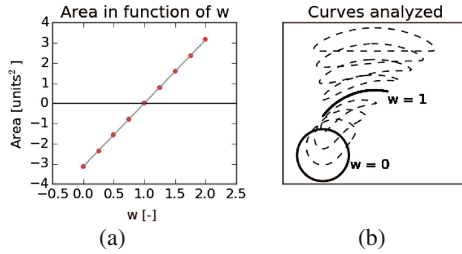
Fig. 9. Variation of area with respect to ratio w

By substitution, we obtain the following expression:

$$Area = \pi \cdot AC^2 \left(CE/CD - 1 \right)$$

The sign of the area tells us if the end effector is travelling clockwise or counterclockwise. Since this information is not known beforehand, we modify the constraint as such:

$$Area = \pi \cdot AC^2 \left| CE/CD - 1 \right| \tag{3}$$

The area is a constant computed from the input curve. Since we can constrain the area of the coupler curve, even smaller sample numbers yield precise solutions. Cases such as seen in Fig. 8 are no longer possible. This allows keeping the model small.

4.5   Simple Design Software

A software application implementing the solving process was developed in Python. It allows the user to draw a curve and returns a four-bar linkage that approximates it. The user draws a curve by positioning control points on a minimal graphic interface as shown in Fig. 10(a). The curve is then analyzed. A few samplings are done with different sample numbers. For each sample number,
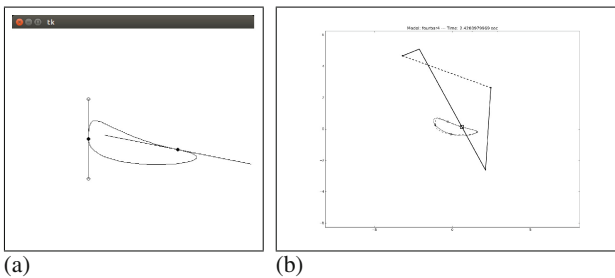
Fig. 10. Design software screenshots; (a) User draws a curve; (b) Matching linkage is displayed.

two models are constructed: one with constraint (1) and the second with constraint (2). A portfolio approach is used and all models are launched in parallel. When a solution is returned, its distance to the input curve is evaluated with Q. If Q is below the user-defined threshold, all processes stop and the best solution is returned and displayed to the user, as shown at Fig. 10(b).

5 Experimentation

We first show a precision and speed comparison with a genetic algorithm. Then, we characterize the performance of our approach. Last, we demonstrate the flexibility of the model by using it to design a robotic gripper.

We use the software described in Sect. 4.5 throughout the experimentation. We generated a benchmark of 100 random curves. For each instance, N different samplings are made. For each sampling, we launch the two possible linkage configurations (constraints (1) or (2)). A total of $2N$ models are solved in parallel. If a solution with Q lower than threshold T is found, the execution is stopped and the solution is returned. Tests conducted with the experimental timeout of 900 s demonstrated that 84.5 % of solutions were returned before 60 s, and 99.6 % were returned before 400 s. Thus, the timeout was set at 400 s. The solving flow is shown on Fig. 11.
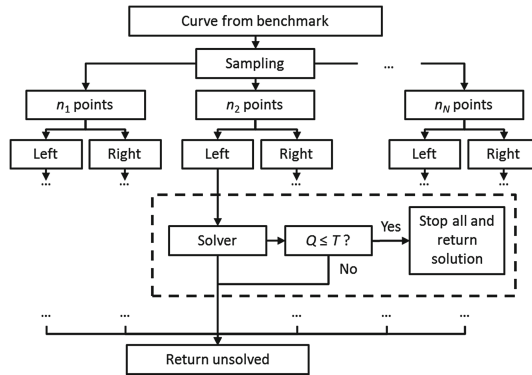
Fig. 11. How a curve is solved

5.1 Benchmark

The benchmark consists of 100 coupler curves of randomly generated linkages. The linkages were generated within the search space of the model. The curves are resized to fit inside a 4 by 4 units square centred at the origin. All curves measure at least 1 unit at their widest. This benchmark spans a wide range of shapes in the search space of our model, which all possess at least one solution. Some curves are presented at Fig. 12.
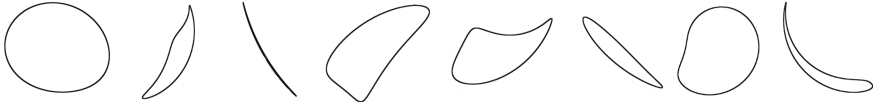
Fig. 12. Example curves from the benchmark

5.2   Results

Comparison with Genetic Algorithm. To present the performance of our non-convex optimization approach, we compare it to results obtained with the genetic algorithm proposed by Cabrera [7], thereafter referred to as the GA.

For the comparison, we replace the *solver* block from Fig. 11 either with the non-convex solver *Couenne* or the GA. The rest of the solving flow remains unchanged. Three samplings are done ($N = 3$) with $n_1 = 6$, $n_2 = 7$ and $n_3 = 8$. The threshold T is set at 5%, so when a solution with lower Q value is returned, the execution stops. We set $s = 0.1$, and $e_\mathrm{u} = 0.01$, which was found to yield the best performance through iterative testing. Figure 13 shows the distribution of the solutions with respect to Q at timeout. The metric quantifies how well the input and output curves match in a continuous way.
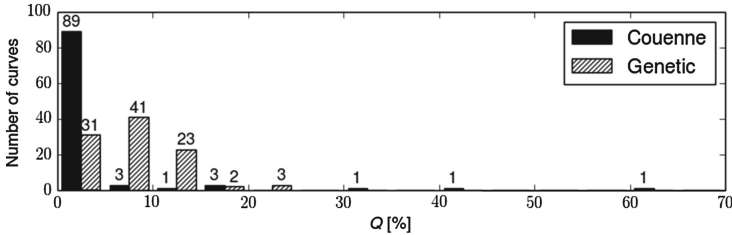
Fig. 13. Distribution of Q values for non-convex optimization and evolutionary approaches

We see that the majority of the curves were solved by Couenne with Q lower than 5%. In contrast, all curves solved by the genetic algorithm returned a Q value below 20%, but less precise on average. Table 3 emphasizes that the median Q returned by Couenne and the median absolute deviation are lower than those of the genetic algorithm.

For the non-convex optimization, Q is computed after Couenne has returned an optimal solution. Therefore, any solution returned by Couenne before timeout is optimal with respect to the discrete metric of the model. As for the GA, Q is computed once every few hundred generations. This constitutes an advantage for the GA because sub-optimal solutions found by Couenne must time out before evaluation. Even so, as shown in Fig. 14, the non-convex optimization approach is faster and times out less often.
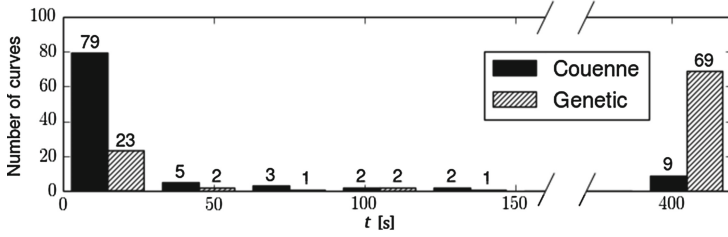
Fig. 14. Distribution of solving times for both approaches. The curves at 400 timed out.

Table 3. Average, variance, median and median absolute deviation of Q.

Approach	$\bar{Q}$	$\sigma^2(Q)$	$\tilde{Q}$	$\mathrm{MAD}(Q)$
Couenne	3.22	71.59	1.00	1.48
Genetic	8.02	16.52	7.25	10.75

Table 4. Number of curves solved under 5, 60 or 400 s with different samplings

Sampling	Area	$Q < 5\%$			No
		5 s	60 s	400 s	solution
{4, 5, 6}	**Yes**	**59**	**83**	**92**	**0**
{4, 5, 6}	No	37	58	63	0
{6, 7, 8}	Yes	51	81	89	1
{6, 7, 8}	No	50	68	78	1
{10, 12, 16}	Yes	30	59	69	11
{10, 12, 16}	No	33	57	66	14

Bounds tightening allows propagation of the restricted domain of variable e. This considerably reduces the search space from the beginning. As for the GA, the final solution depends a lot on the initial random population. Though it consistently finds a reasonable approximation of the curve, it usually stalls in local minima.

Characterization. We show the critical impact of the area constraint and how the feature identification sampling improves the model compared to a uniform sampling.

To evaluate the impact of the area constraint, the benchmark was solved twice over three sample number sets; once with the area constraint and once without. Table 4 shows the number of curves in the benchmark solved with Q lower than 5 % in less than τ seconds, for three values of τ. The number of curves with no solution returned is given.

Higher sample numbers yield longer times of computation without significantly improving the accuracy. In general, the area constraint improved the number of curves solved. Also, when the fewer sampling points are used, the area constraint is most efficient. Without the area constraint, the software performs best with sample numbers $\{6, 7, 8\}$. With the area constraint, lower sample numbers yield a better performance.

The feature identification sampling is compared to a uniform sampling with no analysis of the curve. The experiment was conducted with sets of sample

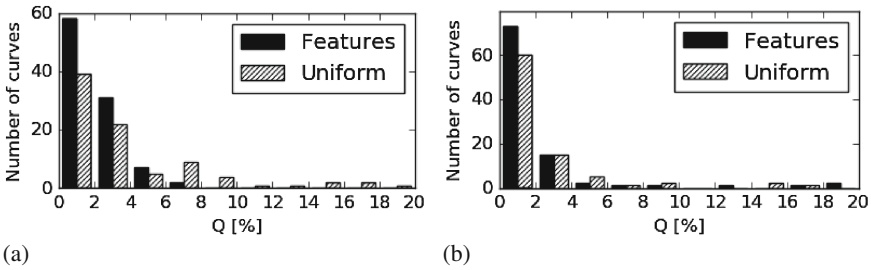numbers $\{4, 5, 6\}$ and $\{6, 7, 8\}$. Figure 15 shows how the sampling affects the distribution of Q.

Fig. 15. Distribution of Qs over benchmark with both sampling techniques; (a) sample numbers $\{4, 5, 6\}$; (b) sample numbers $\{6, 7, 8\}$

For both sets of sample numbers, the feature identification brought the Q distribution closer to 0%. This shows that without increasing the complexity of the model, choosing points strategically can help achieve greater precision.

Design of a Gripper. A benefit of using mathematical optimization is that the model is easily customizable for specific applications. Say we wish to design a gripping mechanism made of symmetric four-bar linkages such that the tip goes through four points, with low precision p_{low} for the first three points and high precision p_{high} on the last. Furthermore, the location of the anchors is restricted. The problem is shown at Fig. 16(a).
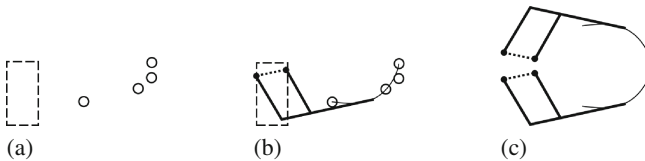
Fig. 16. (a) Target points and anchor bounding box; (b) Synthesized four-bar linkage; (c) Gripper

To adapt the model, only the following modifications need to be done. We replace $A_x, A_y, B_x, B_y \in \{-10, 10\}$ by $A_x, B_x \in \{x_{\mathrm{min}}, x_{\mathrm{max}}\}$; $A_y, B_y \in \{y_{\mathrm{min}}, y_{\mathrm{max}}\}$. We set $e_{\mathrm{u}} = p_{\mathrm{low}}$. We add constraint $(T_{x_3} - E_{x_3})^2 + (T_{y_3} - E_{y_3})^2 \leq p_{\mathrm{high}}$ and disable the area constraint. The resulting gripping mechanism shown at Fig. 16(b) was obtained in $0.20\,\mathrm{s}$, with the modified model.

5.3 Discussion

Our software can quickly and accurately synthesize collinear four-bar linkages for given coupler curves. Indeed, the speeds reached are suitable for interactive applications. Because we use a non-convex optimization solver, our approach is flexible and can be readily adjusted to meet specific needs or different goals. Unlike analytical approaches [19,24], we are not limited by the number of target points to reach. Moreover, the area constraint allows the solver to extrapolate between the target points.

Our method aims at matching a continuous curve rather than a discrete set of target points. However, many related works [6,7] focus on matching target points. Our results show capabilities in both goals. Indeed, the distance to the target point is bounded by the error, which cannot be higher than e_u or to a maximum of 1 % of the size of the curve. Any solution discussed matched its target points at least to this precision.

Our approach also presents benefits compared to machine-learning approaches. A database cannot guarantee coverage of the whole search space. With our approach, the search space is fully explorable and only limited by user-defined restrictions.

Though we focused on minimizing the error, this can be easily changed by replacing the objective function. One could minimize the sum of dimensions, the area of the coupler curve, or the difference of area between the input curve and the output curve.

5.4 Future Work

The software usability could be improved by providing tools to edit constraints.

Our software could extend to four-bar linkages where points $\mathbf{C}$, $\mathbf{D}$ and $\mathbf{E}$ are not collinear. Difficulties include more symmetric configurations and the generalization of the area constraint. Joints such as sliders and complex mechanisms such as geared five and six-bar mechanisms could be modelled. 3D-mechanisms could also be tackled. Our software could be combined with other design analyses such as stress analysis. Multiple linkages could be linked to a gearing software for timing control. Finally, the model could be generated as the user defines his own mechanisms by adding bars and joints.

6 Conclusion

The current state of the art of four-bar linkage synthesis is limited in speed, memory consumption, lack of optimality or lack of generality. Our paper contributes an improved method to the general and fast solving of mechanical linkages. We showed how to accurately solve complete coupler curves in a short time for collinear four-bar linkages using non-convex optimization. For closed curves, a novel constraint can leverage the area of the curve for increased accuracy and performance. For our best model, 90 % of the curves could be solved under 400 s, 59 % of which below 5 s.

Our approach was implemented in simple software where a user can enter a curve and visualize the solution found. This milestone paves the way for modelling mechanisms of increased complexity such as the general four-bar linkage or five-bar linkages. It also provides a very flexible basis for solving four-bar linkages with various constraints or different objectives.

References

1. Ibex library online documentation. http://www.ibex-lib.org/doc/. Accessed 28 June 2016
2. Acharyya, S.K., Mandal, M.: Performance of eas for four-bar linkage synthesis. Mech. Mach. Theor. **44**(9), 1784–1794 (2009)
3. Achterberg, T.: Scip: solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009). http://mpc.zib.de/index.php/MPC/article/view/4
4. Androulakis, I.P., Maranas, C.D., Floudas, C.A.: alphabb: a global optimization method for general constrained nonconvex problems (1995)
5. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**(4–5), 597–634 (2009)
6. Bulatovic, R.R., Djordjevic, S.R.: Optimal synthesis of a four-bar linkage by method of controlled deviation. Theor. Appl. Mech. **31**(3–4), 265–280 (2004)
7. Cabrera, J.A., Simon, A., Prado, M.: Optimal synthesis of mechanisms with genetic algorithms. Mech. Mach. Theor. **37**(10), 1165–1177 (2002)
8. Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R.W., Matusik, W., Bickel, B.: Computational design of mechanical characters. ACM Trans. Graph. **32**(4), 83:1–83:12 (2013)
9. Dijksman, E.A.: Motion geometry of mechanisms. CUP Archive (1976)
10. Granvilliers, L., Benhamou, F.: Algorithm 852: RealPaver: an interval solver using constraint satisfaction techniques. ACM Trans. Math. Softw. **32**(1), 138–156 (2006)
11. Groß, W.: Grundzüge der mengenlehre. Monatsh. für Math. **26**(1), A34–A35 (1915)
12. Hoeltzel, D.A., Chieng, W.-H.: Pattern matching synthesis as an automated approach to mechanism design. J. Mech. Des. **112**(2), 190–199 (1990)
13. IBM. IBM ILOG CPLEX Optimization Studio: High-performance software for mathematical programming and optimization (2016). http://www.ilog.com/products/cplex/
14. Kinzel, E.C., Schmiedeler, J.P., Pennock, G.R.: Function generation with finitely separated precision points using geometric constraint programming. J. Mech. Des. **129**(11), 1185–1190 (2007)
15. Lin, Y., Schrage, L.: The global solver in the lindo api. Optim. Methods Softw. **24**(4–5), 657–668 (2009)
16. Norton, R.L.: Design of Machinery: An Introduction to the Synthesis and Analysi of Mechanisms and Machines. WCB McGraw-Hill (1999)
17. O'sullivan, B., Bowen, J.: A constraint-based approach to supporting conceptual design. In: Gero, J.S., Sudweeks, F. (eds.) Artificial Intelligence in Design 1998, pp. 291–308. Springer, Netherlands (1998)
18. Radhakrishnan, P., Campbell, M.I.: A graph grammar based scheme for generating and evaluating planar mechanisms. In: Gero, J.S. (ed.) Design Computing and Cognition 2010, pp. 663–679. Springer, Netherland (2011)

19. Sandor, G.N., Erdman, A.G.: Advanced Mechanism Design: Analysis and Synthesis. Prentice-Hall, Inc., Englewood Cliffs (1984)
20. Stöckli, F.R., Shea, K.: A simulation-driven graph grammar method for the automated synthesis of passive dynamic brachiating robots. In: ASME 2015 IDETC/CIE Conferences. American Society of Mechanical Engineers (2015)
21. Subramanian, D.: Conceptual design and artificial intelligence. In: Proceedings of IJCAI 1993, pp. 800–809. Morgan Kaufmann Publishers Inc., San Francisco (1993)
22. Subramanian, D., Wang, C.-S.: Kinematic synthesis with configuration spaces. Res. Eng. Des. **7**(3), 193–213 (1995)
23. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**, 225–249 (2005)
24. Uicker, J.J., Pennock, G.R., Shigley, J.E.: Theory of Machines and Mechanisms. Oxford University Press, Oxford (2011)
25. Unruh, V., Krishnaswami, P.: A computer-aided design technique for semi-automated infinite point coupler curve synthesis of four-bar linkages. J. Mech. Des. **117**(1), 143–149 (1995)
26. Zhu, L., Xu, W., Snyder, J., Liu, Y., Wang, G., Guo, B.: Motion-guided mechanical toy modeling. ACM Trans. Graph. **31**(6), 127:1–127:10 (2012)

Using Constraint Programming for the Urban Transit Crew Rescheduling Problem

Xavier Lorca[1,2]([✉]), Charles Prud'homme[1,2], Aurélien Questel[1,2], and Benoît Rottembourg[1,2]

[1] TASC - École des Mines de Nantes, Université de Nantes, Inria,
LINA UMR 6241, Nantes, France
{xavier.lorca,charles.prudhomme,
aurelien.questel,benoit.rottembourg}@mines-nantes.fr
[2] EURODECISION Versailles, Versailles, France
{xavier.lorca,charles.prudhomme,
aurelien.questel,benoit.rottembourg}@eurodecision.com

Abstract. Scheduling urban and trans-urban transportation is an important issue for industrial societies. The Urban Transit Crew Scheduling Problem is one of the most important optimization problem related to this issue. It mainly relies on scheduling bus drivers' workday respecting both collective agreements and the bus schedule needs. If this problem has been intensively studied from a tactical point of view, its operational aspect has been neglected while the problem becomes more and more complex and more and more prone to disruptions. In this way, this paper presents how the constraint programming technologies are able to recover the tactical plans at the operational level in order to efficiently help in answering regulation needs after disruptions.

1 Context and Opportunities

Scheduling urban and trans-urban transportation is an important issue for industrial societies. Several aspects are considered by territorial collectivities and transportation operators: human and material resource for, environmental and social constraint enforcement, user need requirements. Basically, there exists two central problems: transit scheduling (vehicles - buses, trains, tramways - planning on routes) and driver duty planning (assigning crew to those routes). These problems become more and more complex (regulation, network expansion) and more and more prone to *disruptions* (city events, accidents, resource failures) in the operational phase.

This is the context in which the Urban Transit Crew Scheduling Problem (UTCSP) has been introduced [5]. In our proposal, we have to schedule bus drivers' workdays according to several constraints mainly related to: (1) collective agreements, *e.g. breaking rules* (basically, how long a bus driver can work before a break); (2) the bus schedule itself, *e.g. chaining rules* (geographic position, schedule compatibility between two tasks, etc.). From a tactical point of

view, the UTCSP has been thoroughly studied both at the academic and industrial levels. Primarily, the technologies derived from mathematical programming (from integer linear programming to column generation and dynamic programming) dominate the literature and offer satisfying results. The problem is mainly solved using a set covering approach that represents bus schedule as a set of tasks, linked together by chaining rules and respecting the breaking rules. From an operational point of view, these technologies become useless due to their time consumption. Such a pitfall leads the operators to manually repair the tactical solutions after a disruption (at the operational level), and to consequently derive the applied schedule quality.

Focusing on the operational point of view of the UTCSP, *a.k.a.* Urban Transit Crew Rescheduling Problem (UTCRP), this paper presents how the Constraint Programming (CP) technologies are able to recover the tactical plans at the operational level in order to efficiently help answer regulation needs after disruptions. Precisely, it is shown how a constraint model of the UTCSP can be simply modified to address the UTCRP, its operational reformulation.

The paper is composed of six parts. The present one has introduced the context of the UTCSP and has motivated the opportunities for the CP technologies to tackle the rescheduling problem, namely the UTCRP. Next, Sect. 2 is dedicated to the related works and the operational context of the UTCSP. Section 3 is the main section of the paper. A CP model for the UTCSP is first presented. Next, it is shown how simple modifications of the UTCSP lead to a model for the UTCRP. Then, Sect. 4 introduces a common search strategy for UTCSP and its operational version. Section 5 reports the empirical evaluations of both UTCSP and UTCRP on industrial instances. Finally, Sect. 6 concludes.

2   Related Works and Operational Context

The most widely used approach for this problem is related to a set partitioning problem [2]. It is mainly solved by a Branch-and-Bound algorithm where the lower bound is computed through a column generation procedure [4]. In this case, the subproblem corresponds to an Elementary Shortest Path Problem with Resource Constraints (ESPPRC). Most of the time, it is solved using dynamic programming [6,10]. However, according to the number and the nature of the resource constraints, generating a pool of column during a preprocessing may sometimes be more efficient [11]. The resulting Branch-and-Price algorithm is designed to be an exact model which, nevertheless, may fail to optimally solve very large problems (with more than thousands tasks) [3,5,20]. To bypass this issue, a common approach consists in truncating the search tree [8], the lower bound quality is ensured then to be the nearest possible from optimal solutions. Even if many meta-heuristic algorithms have been proposed for the UTCSP [7,12,19,20], the most efficient industrial softwares such as Austrics,[1]

---

[1] http://www.trapezegroup.com.

Hastus,[2] GoalDriver[3] or LP-EasyDriver [4] are based on a truncated version of the exact Branch-and-Price algorithm.

Constraint Programming Attempt. The UTCSP has not been many studied by the CP community. The most related work is [21]. It introduced a pure CP model for a variant of the UTCSP, as well as a hybrid approach composing a column generation algorithm with a CP model dedicated to generate the columns. Nevertheless, the proposed pure CP approach is quite poor because it does not embed global constraints, like REGULAR, to express the regulation needs. Moreover, the search strategy is limited to the *first-fail* principle which is clearly inappropriate. Consequently, not more than 30 tasks can be scheduled to optimality and feasible solutions can be provided with only at most 125 tasks. However, the main contribution is related to the hybridization. Thus, it is shown that the hybrid approach (combining column generation and CP) can be efficient up to 150 tasks.

Operational Context. To the best of our knowledge, there is no literature dealing with the rescheduling of the UTCSP. So, we present an industrial point of view which motivates our proposal. Disruptions can occur a few days before the buses run. Typically, some special event is programmed in the city, like a football match or local roadworks and the bus schedules must be adapted "accordingly". Without loss of generality, tasks to be performed by the buses can either be added, deleted or their duration modified. In some cases buses have connections with trains at the coach station, and a slight change of the train timetable (imposed by French Railways for instance) can have an impact on the bus task at this place in the network. From an operational point of view, a compromise must be found by the planning team: on the one hand, building a new cost-efficient schedule covering the modified tasks and, on the other hand, building a schedule not "too different" from the regular daily schedules. For bus drivers, new workdays might have an impact on either security or comfort, for instance when the new workday ends later than usual.

Today, tools are mainly focussed on finding optimal solutions and tends to produce–when tasks are altered–bran-new schedules that "destroy" the daily ones. In practice, human operators adopt a workaround strategy and manually fix large parts of the schedule, asking the solver tool to optimize only subparts of the schedule. This approach clearly avoids too much perturbation but consequently offer suboptimal solutions. It is also a time consuming effort. The UTCRP consists then in managing a good balance between cost optimality and schedule updates, with dedicated metric and constraints on top on the classical ones. On the converse to pure tactical crew scheduling tools, the computational time of rescheduling must be short, as various trials and errors can have to be experimented by the planning team which is evaluating the impact of what-if scenarios.

---

[2] http://www.giro.ca.
[3] http://www.goalsystems.com.
[4] http://www.eurodecision.fr.

3   Constraint Programming Model

In the UTCSP, tasks have to be assigned to bus drivers. Formally, given n tasks that are to be assigned to at most m bus drivers, the objective is to find a full assignment that minimizes the cost and satisfies breaking rules and chaining rules. A bus driver is mainly characterized by a unique identifier, a skill level, among *novice* and *expert*, and a workday duration and a hourly cost induced by its skill. A novice can only execute low level skill tasks while an expert can execute any type of tasks and can work longer but at a higher hourly cost. The maximal number of novices, *nbNovices* and experts, *nbExperts*, needed to trivially satisfy the problem is determined from the input. A same bus driver can perform many tasks, but one task is only processed by a single bus driver. Each task is defined by its fixed beginning time, B_i, its fixed duration, D_i, its fixed end time, E_i such that, $E_i = B_i + D_i$, $\forall i \in 1..n$. The initially unknown bus driver performing the task is denoted A_i. In addition, a task requires exactly one skill, K_i: either high level skill, only experts can do it, or low level skill, anyone can do it.

A first constraint is that tasks assigned to the same bus driver should not overlap in time. A second constraint deals with the allocation of breaks to each bus driver. This is done by stating a SHIFT constraint, which we now describe.

Definition 1 (SHIFT). *A* shift *is a maximum sequence of tasks assigned to a same bus driver such that the time gap between any two consecutive tasks is shorter than a given threshold minBreak.*

Two consecutive shifts of a bus driver are separated by a *break* of minimum duration *minBreak* and define its workday. Two consecutive tasks of a shift are separated by a gap shorter than *minBreak*. The *span* of a shift is the difference between the end time of its last task and the beginning time of its first task. It is bounded by a given threshold *maxSpan*.

3.1   Modeling the Shift Constraint

Concisely expressing constraints like SHIFT is hard and has been recently studied. In [1], a possible model is presented to manage the SHIFT problem based on the REGULAR [14] and GLOBAL CARDINALITY [17] constraints. We do not report such a model in this paper because of its time generation and memory consumption. Another interesting model, also introduced in [1], addresses most of the pitfalls of the previous one: a STABLEKEYSORT-based model. In such a model, a *decomposition* of the SHIFT constraint is expressed as a conjunction of a STABLEKEYSORT constraint and simple arithmetical and logical constraints. The STABLEKEYSORT(L, P, S, k) constraint is declared with:

– $L = \langle A_i, B_i, D_i, E_i \rangle \mid i \in 1..n]$: a list of task attribute tuples,
– P: an optional permutation list (not required here),

$$\text{SHIFT}([\langle A_i, B_i, D_i, E_i\rangle \mid i \in 1..n], [\langle A'_i, B'_i, D'_i, E'_i, Y_i\rangle \mid i \in 1..n],$$
$$minBreak, maxSpan) \Leftrightarrow (1) \wedge (2) \wedge (3) \wedge (4) \wedge (5) \wedge (6)$$

$$E_i = B_i + D_i, \ \forall i \in 1..n \tag{1}$$

$$\text{STABLEKEYSORT}([\langle A_i, B_i, D_i, E_i\rangle \mid i \in 1..n], [\langle A'_i, B'_i, D'_i, E'_i\rangle \mid i \in 1..n], 2) \tag{2}$$

$$A'_{i-1} < A'_i \ \vee \ E'_{i-1} \le B'_i, \ \forall i \in 2..n \tag{3}$$

$$Y_i = 1 \Leftrightarrow \begin{cases} \textbf{false} & \text{if } i = 1 \\ A'_i = A'_{i-1} & \text{if } i \in 2..n \end{cases} \tag{4}$$

$$X_i = 1 \Leftrightarrow \begin{cases} \textbf{false} & \text{if } i = 1 \\ Y_i \ \wedge \ B'_i - E'_{i-1} < minBreak & \text{if } i \in 2..n \end{cases} \tag{5}$$

$$R_i = \begin{cases} D'_i & \text{if } i = 1 \\ D'_i + X_i \cdot (R_{i-1} + B'_i - E'_{i-1}) & \text{if } i \in 2..n \end{cases} \tag{6}$$

where

$$R_i \in 1..maxSpan, \ \forall i \in 1..n$$

Decomposition 1: Decomposition of the SHIFT constraint [1].

- $S = \langle A'_i, B'_i, D'_i, E'_i\rangle \mid i \in 1..n$: a stable and non decreasing rearrangement of L,
- $k = 2$: number of first positions to consider in the tuples.

Doing so, the STABLEKEYSORT(L, P, S, k) provides a *view* of tasks of L in which tasks are sorted by workdays. Hence, it eases the expression of the required constraints which can be directly expressed on sorted variables, while their expression could be more tedious otherwise.

Decomposition 1 depicts how the SHIFT constraint is expressed using a STABLEKEYSORT constraint in the state-of-the-art. Constraint (1) ensures tasks integrity. Constraint (2) maintains the rearrangement of task attribute tuples, here only A_i and B_i are considered to sort tuples. Constraint (3) ensures that two consecutive tasks either belong to the different workdays or are chronologically ordered in the same workday and thus do not overlap in time. Constraints (4) and constraint (5) introduce auxiliary 0..1 variables: Y_i indicates whether two consecutive tasks $i - 1$ and i are in the same workday, X_i indicates if two tasks $i - 1$ and i are in the same shift. Finally, a last set of auxiliary 1..$maxSpan$ variables R_i are needed in constraint (6), they compute the shift length up to the end of task i.

3.2   A Constraint-Based Model for the UTCSP

The targeted problem is based on a central SHIFT constraint but also comes with the following additional variables and constraints, presented in Model 1. For modeling purpose, bus drivers whom unique identifier is in 1..$nbNovices$ are novices, those with greater identifier, up to $nbNovices + nbExperts$, are experts. Hence, constraint (8) introduces the auxiliary 0..1 variables K_i which indicate whether the bus driver performing the task i is an expert. P_i, W_i, C_i denote

Minimize

$$\sum_{i=1}^{n} C_i \qquad (7)$$

subject to

$$\textsc{Shift}([\langle A_i, B_i, D_i, E_i \rangle \mid i \in 1..n], [\langle A'_i, B'_i, D'_i, E'_i, Y_i \rangle \mid i \in 1..n], minBreak, maxSpan)$$

$$K_i = 1 \Leftrightarrow A'_i > nbNovices, \ \forall i \in 1..n \qquad (8)$$

$$P_i = \begin{cases} 0 & \text{if } i = 1 \\ B'_i - E'_{i-1} & \text{if } i \in 2..n \end{cases} \qquad (9)$$

$$Y_i \Rightarrow P_i \leq maxSpan, \ \forall i \in 2..n \qquad (10)$$

$$W_i = \begin{cases} D'_i & \text{if } i = 1 \\ D'_i + X_i \cdot (W_{i-1} + P_i) & \text{if } i \in 2..n \end{cases} \qquad (11)$$

$$C_i = \begin{cases} (1 - Y_{i+1}) \cdot cost_{K_i, W_i} & \text{if } i \in 1..n-1 \\ cost_{K_i, W_i} & \text{if } i = n \end{cases} \qquad (12)$$

Redundant constraint

$$\textsc{AllDifferent}([A_k \mid k \in clique(A)]) \qquad (13)$$

where $\forall i \in 1..n$,

$$P_i \in -dayDuration..dayDuration, \ W_i \in 0..dayDuration, \ C_i \in 0..dayDuration \times costExpert,$$

Model 1: Formulation of the Urban Transit Crew Scheduling Problem.

respectively the period between two consecutive tasks $i-1$ and i, stated by constraint (9), the duration of a shift including task i, stated by constraint (11), and the cost of a bus driver's workday, stated by constraint (12). Note that the workday duration is bounded thanks to that last constraint. It is expressed with a TABLE [9] constraint wherein possible combinations of skill and workday and costs are listed. Constraint (10) ensures that, in a shift, the period between two tasks $i-1$ and i is less than or equal to the *maxSpan*. Based on an *a priori* analysis of the tasks network, AllDifferent [16] constraints (13) make sure that tasks belonging to the same clique are performed by different bus drivers.

Finally, the model is improved by considering symmetries and their related symmetry breaking constraints, depicted by Model 2. First, we introduce M_i^R and M_i^E which maintain, for each skill, the list of identifier of used bus drivers (constraints (14) and (15)), they also help to count the number of bus drivers per skill. Then, constraints (16), (17), (18) and (19) break symmetries.

3.3   A Constraint-Based Model for the UTCRP

In order to stay close to the operational context of the bus networks, we have built instances for the UTCRP based on disruptions of UTCSP instances. Indeed, a daily schedule is considered as known, and the tasks' disruptions occur locally. So, the disruptions can be easily simulated from the UTCSP instances. We have to keep in mind that these disruptions can either be:

$$M_i^R = \begin{cases} A_i' & \text{if } K_i = 0 \\ 0 & \text{if } K_i = 1 \wedge i = 1 \\ M_{i-1}^R & \text{if } K_i = 1 \wedge i \in 2..n \end{cases} \tag{14}$$

$$M_i^E = \begin{cases} A_i' & \text{if } K_i = 1 \\ nbNovices & \text{if } K_i = 0 \wedge i = 1 \\ M_{i-1}^E & \text{if } K_i = 0 \wedge i \in 2..n \end{cases} \tag{15}$$

$$M_{i-1}^R \leq M_i^R, \ \forall i \in 2..n \tag{16}$$

$$M_{i-1}^E \leq M_i^E, \ \forall i \in 2..n \tag{17}$$

$$\text{IntValuePrecedeChain}(A, [j \mid j \in 1..nbNovices]) \tag{18}$$

$$\text{IntValuePrecedeChain}(A, [j \mid j \in nbNovices..nbNovices + nbExperts]) \tag{19}$$

where $\forall i \in 1..n$,

$$M_i^R \in 1..nbNovices, \ M_i^E \in nbNovices..nbNovices + nbExperts$$

Model 2: Improving Model 1 with symmetry breaking constraints.

– Creation of new tasks: the bus line has to serve the stadium station at 1 o'clock in the morning, due to a rock concert;
– Task deletion: a roadwork prevents the bus to use the road and stop at the station;
– Duration change: roadworks slow down the traffic;
– Start time change: a train timetable has been modified, and the connection forces a change of the bus departure time.

Consequently, UTCRP aims at producing new schedules which: (a) cover all the tasks (previous and new); (b) do not differ too much in terms of cost from the initial daily schedule; and (c) ensure that the workday modifications are minimal for the bus drivers. These schedules have to be produced within a few seconds of computational time so that the planning team can test different scenarios and choose the most convenient one.

From a constraint programming point of view, Model 3 presents how to turn the UTCSP model, depicted in Sect. 3.2, into UTCRP. Given I_s, an assignment of n tasks to m bus drivers, let us denote $C^\star$ its cost. Consider that some of the tasks have been disrupted and that I_s has to be repaired. First, the previous objective function, equation (7) in Model 1, is turned into the hard constraint (21): new solutions have to be at most $\epsilon\%$ above $C^\star$. Then, for each set of tasks performed by the same bus driver in I_s, the number of bus drivers needed to perform the same tasks anew is maintained by constraint (22). Finally, the objective function (20) aims at computing solutions similar to I_s in term of unmodified workdays.

4   Search Strategy

A constructive search strategy is defined to dive to a first solution quickly without failure. The A_i variables are selected in lexicographic order. The bus driver that

Minimize
$$\sum_{\ell=1}^{m} N_\ell \tag{20}$$

subject to
$$\sum_{i=1}^{n} C_i < (C^\star \cdot \epsilon)/100 \tag{21}$$

$$\text{NVALUES}([A_\ell \mid \ell \in workday(A)], =, N_\ell) \tag{22}$$

where $\forall \ell \in 1..m$,
$$N_\ell \in 0..n$$

Model 3: Modifications to bring to the CP model of UTCSP to turn it into UTCRP.

performs a task A_i is computed as follow. Bus drivers already performing at least one task are considered first. Those whose workday is not directly compatible with the task to assign (*w.r.t.* either break rules or chaining rules cannot be satisfied) are ruled out. Remaining ones are then tried sequentially. Some more tries are finally considered, allowing the addition of at most one bus driver per type of skill required.

Next solutions are obtained by plugging Large Neighborhood Search (LNS) [18] in, a straightforward two-phase local search-like approach. It partially *relaxes* a given solution and tries to *repair* it. Given an input solution, the relaxation phase builds a *partial solution*: some variables are selected to be relaxed to their initial domain, while the other ones are assigned to their value in the solution. The reparation phase tries to extend the partial solution to a complete one that improves the objective function.

The efficiency of LNS lies in the way variables are selected to be relaxed. In our case, it tries to find a better solution by locally rearranging bus drivers' workday. So, the workdays are first extracted from the A variables in a solution. Then, up to $\theta \in [\![2,4]\!]$ workdays are randomly selected to be rearranged in such way that they overlap in time at least one of the other selected one. The corresponding A variables are relaxed, the other ones are assigned to the same bus driver as declared in the solution. To consolidate even more the partial solution, bus drivers' identifier corresponding of fixed A variables are removed from relaxed A variables' domain. The process is completed with a fast restart strategy [13], which limits the reparation phase to $2n$ failures and avoids spending too much time in hard-to-repair partial solutions.

The same search strategy is applied to both the UTCSP and the UTCRP. The strategy was initially designed to produce few dense workdays, with a local reasoning. When the objective changes to repair assignments, the model is constrained enough to guide the process towards workdays similar to the initial ones.

5 Practical Experiments

For an empirical evaluation, we instantiate the UTCSP model of Sect. 3.2. The SHIFT constraint holds for a 15-min *minBreak* and 2-h *maxSpan*. A novice cannot work more than 8 h and its hourly cost is fixed to 10. An expert cannot work more than 9 h, and its hourly cost is fixed to 17. We consider here the following additional rule: any working bus driver has a minimal 6-h pay, even if its workday lasts for less than 6 hours. This constraint is directly encoded into constraint (12). Finally, the UTCRP model depicted in Sect. 3.3 also instantiates ϵ, the distance to $C^{\star}$, to 10.

There are two sets of instances. A first set aims at comparing the constraint-based approach with the state-of-the-art one on the UTCSP. It is composed of real-world problem instances involving up to 3,200 tasks. A second set aims at evaluating how repairing a disrupted assignment is made easy with a constraint-based approach using the UTCRP (Model 3). First of all, the best solution found by EURODECISION for the instance with 800 tasks is selected, it is composed of 160 workdays. Then, this instance is disrupted applying the following process: a task j is randomly selected to be removed. Starting from j, within a range of $\beta \in \{30, 60, 90, 120\}$ minutes before B_j and after E_j, tasks on a path to j, that is *w.r.t.* breaking rules and chaining rules, are removed. Tasks that overlap the range in time are reduced (either B_i or E_i is modified). The selection-and-removing phase is repeated $\alpha \in [\![1..5]\!]$ times. This results in a set of 20 instances to be repaired with up to 25 % disrupted tasks, which corresponds to our real life context.

Protocol. The experiments were run on a Mac Pro with 8-core Intel Xeon E5 at 3 GHz under MacOS 10.11.3 and Java 1.8.0_25. Each instance of the first set was run with a 2-h limit on its own core, and each instance of the second set was run with a 1-min limit on its own core for CP approach. LP-EasyDriver was evaluated in a 2-h limit for both sets. All of them were run with up to 4 GB of memory. The tools used are: Choco [15] for the constraint programming part, while EURODECISION provides LP-Taskplanner, a generic framework for ESPPRC based column generation models, relying on CPLEX 12.6.2. LP-Taskplanner is embedded in LP-EasyDriver: the latter handles every business aspects while the former deals with the optimization parts. In the following, this tool will be referred to as LP-EasyDriver.

5.1 Solving the UTCSP

In this section, we compare the constraint-based approach for the UTCSP with the one being used by EURODECISION and introduced in Sect. 2. The results are given in Table 1 which is divided into four parts. The first part indicates the instance size n. The second part reports information related to the constraint-based approach: the time to get the first solution (**time**, in seconds), its cost (**first**) and the best cost obtained in 120 min (**best@120**). The third part is about LP-EasyDriver: the time (**time**, in seconds) to compute the lower bound

(**LB**) and a first upper bound (**UB**) then the best upper bound in 120 min (**UB@120**). Finally, the last part reports, when possible, the ratio

$$g(a, b) = \frac{a}{b} \times 100$$

where, here, $a = (best@120 - UB@120)$ and $b = UB@120$. In addition, "*N/A*" denotes that the information is not available or applicable, "*OOT*" denotes that a particular approach runs out of time.

Table 1. Empirical results on the UTCSP: time (**time**, in seconds) for finding the first solution (**first**) and the best solution found in 120 minutes (**best@120**), and for finding the root lower bound (**LB**), the first upper bound (**UB**), and the best upper bound in 120 minutes (**UB@120**). The ratio of best@120 to UB@120 is also reported (**Gap**, in %).

n	Constraint-based approach			LP-EasyDriver					Gap
	time	first	best@120	time	LB	time	UB	UB@120	
200	0.97	5005.00	4352.75	9	4069.81	10	4150.25	4073.75	6.85
400	1.71	9131.75	8318.50	134	7654.78	140	8790.75	7677	8.36
600	2.65	13566.00	12218.25	499	10845.6	4152	11328.5	11014	10.93
800	3.80	17479.25	16375.00	1265	14472.8	23128	14690.2	N/A	N/A
1000	5.51	20857.75	20211.25	2593	17868	OOT		N/A	N/A
1400	9.42	29210.75	28031.00	10245	24214.2	OOT		N/A	N/A
1800	15.22	37394.50	36431.00	33698	31578.2	OOT		N/A	N/A
2200	22.42	44849.50	43856.50	OOT		OOT		N/A	N/A
2600	31.94	52885.75	52011.25	OOT		OOT		N/A	N/A
3000	45.09	59138.50	58656.25	OOT		OOT		N/A	N/A
3200	52.60	65382.75	64584.75	OOT		OOT		N/A	N/A

We observe that:

- The truncated Branch-and-Price approach fails at solving large problems. This observation confirms the state-of-the-art (Sect. 2). The bounds produced are very sharp considering a 2-h time limit. Indeed, such a time limit is below from what is commonly allocated at the tactical level.
- The constraint-based approach is able to find a first solution for any problem size. Moreover, the ability of the model to scale up combined with a constructive search strategy makes possible to provide a first solution in a very short time. Nonetheless, the quality of the best solution found in the given time limit tends to decrease when the instance size n becomes larger. In the end, without any evaluation–neither propagation–of the lower bound, the approach fails at proving optimality.

At a tactical level, where the run time matters less than the quality of the bounds, and up to mid-size problems, the state-of-the-art approach is suitable. On any-size problems, the constraint-based approach is responsive, yet less accurate. At an operational level, when rescheduling tasks is needed, responsiveness becomes indispensable. This is the point we want to evaluate in Sect. 5.2.

5.2   Solving the UTCRP

When a disrupt occurs, its extent is measurable thanks to three indicators: the number of removed or modified tasks (n'), the number of bus drivers whose workday is disrupted (m') and the total number of affected tasks, gathering all tasks of disrupted workdays (n''). In practice, when dealing with rescheduling, not all the tasks are set as input of the model depicted in Sect. 3.3. Indeed, only affected tasks are considered. The remaining workdays are immutable. Recall that the solution selected to be disrupted was made of $n = 800$ tasks and $m = 160$ bus drivers (or workdays).

We report the results of constraint-based approach for the UTCRP. The results are given in Table 2 which is composed of four parts. The first part indicates the disruption parameters β and α and the indicators of its extent n', m' and n''. The second part reports information related to the constraint-based approach: the time to get the first solution (**time**, in seconds) and **gap@first** that is the ratio $g(m - m' + m^1, m)$ where m^1 is the number of workdays covering the n'' tasks (recall that the unaffected workdays are kept immutable). This qualifies the distance from the first solution found to the initial one in terms of number of modified workdays needed to cover the affected tasks. The ratios based on the best solution found in ten seconds (**gap@10**) and 60 s (**gap@60**) relative to the initial solution are also reported. Then, the cost of the best solution found in 60 seconds is indicated (**cost@60**). For indication purpose, the results found with LP-EasyDriver when solving the problems from scratch, without any time limit, are reported in the third part (**best**). Finally, the ratio $g(cost@60 - best, best)$ can be observed in the last part (**Gap**). In addition, "-" denotes that no better solution was found in the elapsed time.

Our observations are threefold:

– The constraint-based approach is responsive as expected. Its ability to quickly reschedule disrupted workdays does not depend on the extent of the disruption. Nevertheless, the quality of the reparation is related to the total number of the affected tasks n''.
– The search strategy is able to compute good quality solutions in term of number of workdays. When less than 83 tasks are affected by the disruption, ten seconds are enough to provide solutions at most 5 % above the number of workdays needed before the disruption. When more tasks are affected, sixty seconds are always enough to find solution at most 5 % above the initially required number of workdays.
– The search strategy is able to compute good quality solution in terms of cost. The best solutions found with CP are very close to the ones computed from

Table 2. Empirical results on the UTCRP: range of the disruption (β) along a path involving a randomly selected task, number of randomly selected tasks (α), number of directly affected tasks (n') and bus drivers (m') and total number of affected tasks (n''), time (**time**, in seconds) for finding the first solution, the ratio of the number of workdays needed in, respectively, the first solution (**gap@first**, in %), the solution found in 10 seconds to (**gap@10**, in %) and the solution found in 60 seconds (**gap@60**, in %) to the initial number of workdays required, the cost of the solution found in 60 seconds (**cost@60**), and the ratio of cost@60 to the best solution (**best**) computed from scratch with the LP-EasyDriver (**Gap**, in %).

Inst. parameters					Constraint-based approach					LP-EasyDriver	Gap
β	α	n'	m'	n''	time	gap@first	gap@10	gap@60	cost@60	best	
30	1	3	3	15	0.85	1.87	0.00	-	14489.50	14462.00	0.19
	2	7	7	39	0.98	4.37	0.62	-	14584.75	14482.50	0.71
	3	11	10	63	1.07	15.62	2.50	0.62	14576.25	14440.00	0.94
	4	15	14	92	1.16	27.50	20.62	3.75	14597.50	14369.25	1.59
	5	18	17	114	1.25	35.62	31.87	4.37	14729.75	14336.50	2.74
60	1	4	4	21	0.86	2.50	0.62	-	14591.50	14446.00	1.01
	2	9	9	41	0.93	4.37	0.62	-	14532.00	14416.00	0.80
	3	15	14	72	1.06	10.62	3.12	0.62	14578.00	14373.50	1.42
	4	18	16	83	1.09	13.12	2.50	0.62	14497.25	14314.50	1.28
	5	22	18	99	1.11	18.75	11.87	0.62	14539.75	14345.00	1.36
90	1	6	6	35	0.91	3.75	0.62	-	14576.25	14464.00	0.78
	2	13	12	76	1.07	13.75	2.50	1.25	14623.00	14466.75	1.08
	3	16	14	84	1.09	16.25	5.62	1.87	14631.50	14424.00	1.44
	4	24	21	125	1.24	29.37	23.12	1.25	14669.25	14434.25	1.63
	5	32	28	168	1.37	42.50	41.25	2.50	14669.75	14294.00	2.63
120	1	8	7	41	0.94	6.25	1.25	-	14563.50	14458.50	0.73
	2	18	15	103	1.18	29.37	19.37	0.62	14571.25	14423.00	1.03
	3	27	20	134	1.31	38.12	31.87	0.62	14551.00	14334.25	1.51
	4	37	28	185	1.39	55.62	53.12	3.75	14743.25	14319.00	2.96
	5	44	33	209	1.45	63.75	61.25	3.75	14722.50	14259.50	3.25

scratch by LP-EasyDriver. Nevertheless, the EURODECISION approach fails at maintaining the workdays which are not impacted by disruptions. Consequently, such approach does not fit the needs of the planning teams.

6   Conclusion and Further Works

In the context of scheduling bus drivers' workdays, this paper first described a constraint-based model for the UTCSP. We empirically confirmed that this CP model scaled up to 3,200 tasks and was able to quickly build good quality solutions. Nevertheless, due to a lack of any lower bound integration, it struggled to improve them enough to reach and prove optimality. Second, we highlighted CP flexibility by turning the UTCSP into its operational counterpart, the UTCRP. The ability of the constraint-based approach to quickly reschedule workdays has been assessed on randomly disrupted instances. Indeed, the closeness of the new scheduling to the initial one in terms of both workdays and cost bears out that CP is a serious runner for that problem.

The responsiveness and the good quality of the solutions found by the CP approach for UTCRP offer new perspectives and advocates for field experiments. EURODECISION will launch pilots with groups of customers in order to measure the ease of use of repaired solutions together with the real life metrics used by the planning teams. Our goal is to reduce the hassle imposed to the planners by the disruptions without loosing too much cost compared to optimality.

References

1. Beldiceanu, N., Carlsson, M., Flener, P., Lorca, X., Pearson, J., Petit, T., Prud'Homme, C.: A modelling pearl with sortedness constraints. In: Gottlob, G., Sutcliffe, G., Voronkov, A., (eds.), GCAI 2015, Global Conference on Artificial Intelligence, vol. 36. EPiC Series in Computing, pp. 27–41. EasyChair (2015)
2. Caprara, A., Toth, P., Fischetti, M.: Algorithms for the set covering problem. Ann. Oper. Res. **98**(1–4), 353–371 (2000)
3. Chen, S., Shen, Y.: An improved column generation algorithm for crew scheduling problems. J. Inf. Comput. Sci. **10**(1), 175–183 (2013)
4. Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.): Column Generation, vol. 5. Springer Science & Business Media, Heidelberg (2006)
5. Desrochers, M., Soumis, F.: A column generation approach to the urban transit crew scheduling problem. Transp. Sci. **23**(1), 1–13 (1989)
6. Feillet, D., Dejax, P., Gendreau, M., Gueguen, C.: An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks **44**(3), 216–229 (2004)
7. Forsyth, P., Wren, A.: An ant system for bus driver scheduling (1997)
8. Franck, B., Neumann, K., Schwindt, C.: Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. OR-Spektrum **23**(3), 297–324 (2001)
9. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Data structures for generalised arc consistency for extensional constraints. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, 22–26 July 2007, Vancouver, British Columbia, Canada, pp. 191–197. AAAI Press (2007)
10. Irnich, S., Desaulniers, G., et al.: Shortest path problems with resource constraints. Column Gener. **6730**, 33–65 (2005)
11. Jacquet-Lagrèzel, É.: Horaires de chauffeurs de bus. Gestion de production et ressources humaines: méthodes deplanification dans les systèmes productifs, p. 287 (2005)
12. Lourenço, H.R., Paixão, J.P., Portugal, R.: Multiobjective metaheuristics for the bus driver scheduling problem. Transportation Sci. **35**(3), 331–343 (2001)
13. Perron, L.: Fast restart policies and large neighborhood search. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (2003)
14. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 482–495. Springer, Heidelberg (2004)

15. Prud'homme, C., Fages, J.-G., Lorca, X.: Choco Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S (2015)
16. Régin, J.-C.: A filtering algorithm for constraints of difference in CSPs. In: Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, vol. 1, pp. 362–367. AAAI Press/The MIT Press (1994)
17. Régin, J.-C.: Generalized arc consistency for global cardinality constraint. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 1996, IAAI 1996, Portland, Oregon, 4-8 August 1996, vol. 1, pp. 209–215. AAAI Press/The MIT Press (1996)
18. Shaw, Paul: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, Michael J., Puget, Jean-François (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
19. Shen, Y., Kwan, R.S.K.: Tabu search for driver scheduling. In: Voß, S., Daduna, J.R. (eds.) Computer-Aided Scheduling of Public Transport, vol. 505, pp. 121–135. Springer, Heidelberg (2001)
20. Silva, G.P., Reis, A.F.S.: A study of different metaheuristics to solve the urban transit crew scheduling problem. J. Transport Lit. **8**(4), 227–251 (2014)
21. Yunes, T.H., Moura, A.V., de Souza, C.C.: A hybrid approach for solving large scale crew scheduling problems. In: Pontelli, E., Santos Costa, V. (eds.) PADL 2000. LNCS, vol. 1753, pp. 293–307. Springer, Heidelberg (2000)

Optimizing Shortwave Radio Broadcast Resource Allocation via Pseudo-Boolean Constraint Solving and Local Search

Feifei Ma[1(✉)], Xin Gao[2], Minghao Yin[2], Linjie Pan[1], Jiwei Jin[3], Hai Liu[1], and Jian Zhang[1]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
maff@ios.ac.cn, panlinjie1993@163.com, stuliuhai@gmail.com
[2] College of Computer Science, Northeast Normal University, Changchun, China
{gaox819,ymh}@nenu.edu.cn
[3] Shan Dong Jiaotong University, Jinan, China
jinjw@ios.ac.cn

Abstract. Shortwave radio broadcasting is the principal way for broadcasting of voice in many countries. An important problem in shortwave radio broadcasting is how to allocate transmission devices to radio programs, so that all radio programs are broadcasted properly and the overall broadcasting effect is optimized. The broadcasting effect of a program is determined by various factors, such as time, location, and device parameters. There are also restrictions on the usage of transmission devices. In this paper, we describe the allocation of shortwave radio broadcast resources as a constrained optimization problem and prove that it is NP-hard. A Pseudo-Boolean constraint formulation for the problem is presented. We also propose an efficient local search algorithm to solve the problem. Both methods are evaluated using real data. Experimental results suggest that we can find an allocation plan with good broadcasting effect quite efficiently.

1 Introduction

Shortwave radio is a kind of radio transmission using shortwave frequencies, ranging from 2 to 30 megahertz (MHz). Such radio waves can be reflected or refracted back to Earth from the ionosphere, allowing communication around the curve of the Earth. Therefore, shortwave radio is very effective for long distance communication. In many countries, shortwave radio broadcasting remains the principal way for broadcasting of voice and music.

An important problem arising in shortwave radio broadcasting is how to assign radio transmission resources to programs, so that the broadcasting quality is optimized. Generally, in a large country like China, there can be thousands of radio transmission devices distributed in dozens of shortwave radio stations all over the territory. For a radio program, the broadcasting effect in its target area may vary from one device to another. It is demanded that each radio program

should be transmitted with a proper transmission device so that the broadcasting satisfies certain criteria in the target area. Meanwhile, it is desirable to maximize the overall broadcasting effect.

In China, the state-level TV programs and radio programs are coordinated by the State Administration of Press, Publication, Radio, Film and Television (SAPPRFT[1]). Currently, the radio programs are managed by staff members manually. This is not efficient, and it is also error-prone. Since the search space of the problem is extremely huge, the staff members have no choice but to rely heavily on previous allocation plans, which are becoming obsolete with the change of programs and devices.

For the last few years, we have been cooperating with the Division of Radio Frequency Assignment of SAPPRFT, to increase the degree of automation in their daily work. Our aim is to design and implement a system that can automatically produce an optimal plan that allocates available transmission devices to the radio programs to be broadcasted. Basically, there are two kinds of tasks: the seasonal allocation and the daily allocation. They only differ in the number of the programs. Seasonal allocation needs to make arrangements for nearly a hundred programs, while daily allocation only handles several programs. So in this paper, we do not distinguish these two tasks. The challenges include, among others, (1) The optimal plan is to be made based on the broadcasting effect data of all possible allocations. Since there are thousands of devices and nearly a hundred radio programs, the search space of the problem is enormous. In fact, the raw data files produced by the propagation software for predicting broadcasting effect amount to hundreds of gigabytes (GB). (2) The problem is time-critical. Especially when some emergency occurs, the system should be able to adapt the allocation plan quickly so that all programs can be transmitted without pause.

The shortwave radio broadcast resource allocation problem (SRBRA) addressed in this paper is derived from our project. In order to focus on the algorithmic aspect, we omit in this paper the minor issues such as data processing and frequency selection which are actually laborious. We formulate shortwave radio broadcast resource allocation as a constrained optimization problem and prove that it is NP-hard. To solve the problem, we propose two complementary approaches, one is based on Pseudo-Boolean Optimization (PBO) and the other is an efficient local search algorithm with quick consistency checking mechanism and the metaheuristic of Greedy Randomized Adaptive Search Procedure [3,12].

The paper is organized as follows. We first describe the shortwave radio broadcast resource allocation problem and prove its NP-hardness in Sect. 2. In Sect. 3, we present a Pseudo-Boolean formulation for this problem. In Sect. 4 we describe the local search algorithm. We evaluate the proposed approaches on real data in Sect. 5. In Sect. 6 we discuss some related works. Finally we conclude the paper.

---

[1] http://www.sapprft.gov.cn/.

2 Problem Description

2.1 Background

Suppose there are m radio programs to be broadcasted by n radio transmission devices. We denote the ith ($1 \leq i \leq m$) radio program by P_i and the jth ($1 \leq j \leq n$) transmission device by D_j. We also denote the set of radio programs by $\mathcal{P}$ and that of transmission devices by $\mathcal{D}$. Program P_i is to be broadcasted to its target area R_i during a predetermined time span $[t_i, t_i']$. In each target area, there are a number of monitoring sites. Note that two programs P_i and P_k may have the geographically same target area, but we still recognize R_i and R_k as separate areas in our problem. The reason is that P_i and P_k are broadcasted on different frequencies, so the broadcasting effect of one program would not interfere with that of the other. A transmission device consists of a transmitter and an antenna. Two different transmission devices D_j and D_k may share the same transmitter or antenna, or the same electric switch in their circuits, hence cannot be used simultaneously. Such devices are called **conflicting devices**, denoted by $conflict(D_j, D_k)$.

A program can only be transmitted with one device. The device D_j, once occupied by program P_i, cannot transmit other programs during the time span $[t_i, t_i']$. Without loss of generality, we use $< P_i, D_j >$ to represent the allocation of transmission device D_j to program P_i. The broadcasting effect of $< P_i, D_j >$ at a monitoring site in R_i is measured by field strength and circuit reliability of the shortwave radio, both of which can be computed with a generic propagation program such as VOACAP [7] or REC533 [2]. The input to the propagation program includes the broadcasting time of P_i, parameters of D_j, the radio frequency, and the locations of the shortwave radio station and the monitoring site.

The broadcasting effect at a monitoring site is considered to be acceptable by the Division of Radio Frequency Assignment of SAPPRFT if the field strength is above 38 dB. The site is qualified if the field strength is above 55 dB and the circuit reliability is above 70 %. For an allocation $< P_i, D_j >$ to be admissible, if at least 60 % of the sites are acceptable. The optimization goal is to maximize the total number of qualified sites in the target areas of all programs.

To model an SRBRA problem, we need to gather a lot of information in advance. Data processing is an important module in our project, as is demonstrated in Fig. 1. As a preprocessing step, we employ a propagation program (REC533) to calculate the field strength and circuit reliability data for each allocation $< P_i, D_j >$ at every monitoring site in R_i, Then we sift out the admissible allocations. For each allocation $< P_i, D_j >$, the number of qualified sites in R_i, denoted as $N_{<P_i, D_j>}$, is also derived. Besides, for each program we find the best frequency to transmit according to certain criterion. In addition to broadcasting effect modeling, we also need to derive program information and device information. In particular, we generate the device conflicting constraints using a conflict checking algorithm. The information obtained from the data processing module is passed to the allocation algorithm, which is the main topic of this paper.
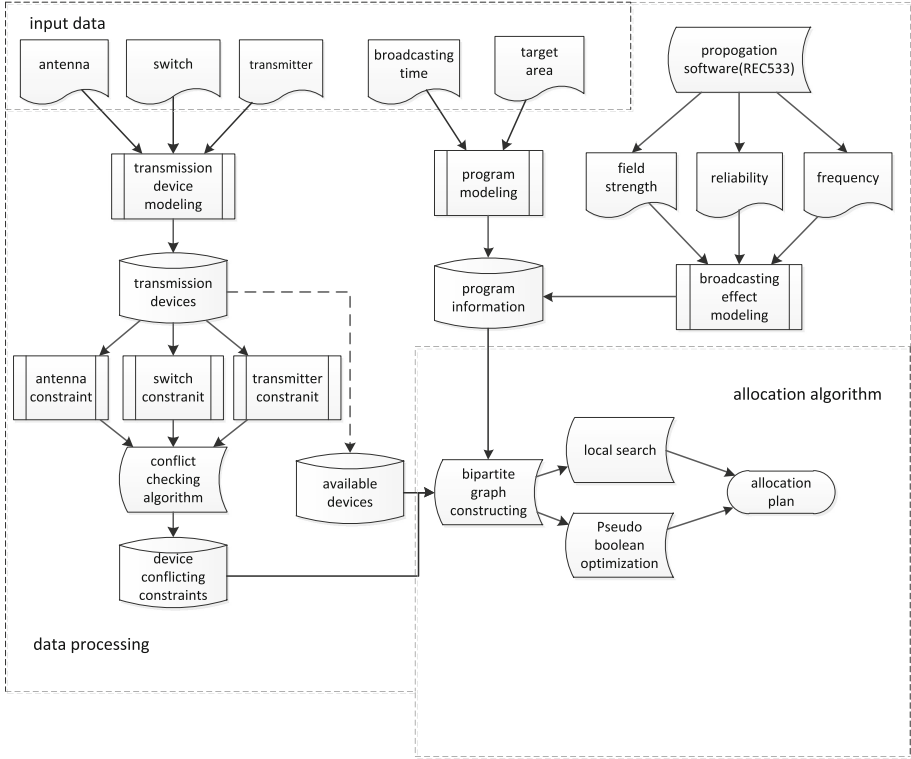
Fig. 1. The architecture of the SRBRA project

2.2 Shortwave Radio Broadcast Resource Allocation

Formally, the SRBRA problem can be defined in the following way.

Definition 1 (Shortwave Radio Broadcast Resource Allocation (SRBRA)). *Given m radio programs and n transmission devices, for each program P_i select a device D_j such that:*

1. *The allocation $< P_i, D_j >$ is admissible.*
2. *If program P_i and program P_k clash in the broadcasting time, i.e., $t_i < t'_k \wedge t_k < t'_i$, then they cannot be transmitted with the same device, or with two conflicting devices.*
3. *The total number of qualified sites $\Sigma_i N_{<P_i, D_j>}$ is maximized.*

The collection of the selected allocations is a solution to the problem, denoted by $\mathcal{S}$. The objective value corresponding to $\mathcal{S}$ is denoted by $N_{\mathcal{S}}$.

The constraint structure of an SRBRA problem can be represented as a variant of weighted bipartite graph, with $\mathcal{P}$ and $\mathcal{D}$ being two disjoint sets of vertices. If the allocation $< P_i, D_j >$ is admissible, there is an edge with weight

$N_{<P_i, D_j>}$ connecting the corresponding vertices. In illustration, we use a solid line to represent such an edge. Moreover, a dashed line is introduced for two programs clashing in the broadcasting time, or for two conflicting devices. The resulting graph is bipartite with respect to the solid lines.

Example 1. Figure 2 demonstrates the weighted bipartite graph of an SRBRA problem with 3 radio programs and 5 transmission devices. The number of qualified sites produced by each admissible allocation is labeled on the corresponding edge. The dashed line connecting P_2 and P_3 suggests that P_2 and P_3 clash in the broadcasting time. The dashed line between D_2 and D_3 suggests that they are conflicting devices. Similarly, we have $conflict(D_4, D_5)$. An optimal solution for this problem is $\{< P_1, D_2 >, < P_2, D_2 >, < P_3, D_5 >\}$. Accordingly, the optimal objective value is 23. Note that P_1 and P_2 can be transmitted with the same device because there is no clash in the broadcasting time.
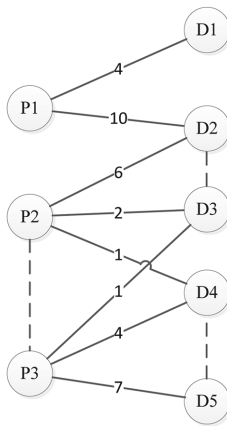
Fig. 2. An example of the SRBRA problem

At first glance, the SRBRA problem resembles the maximum weighted bipartite matching problem. The latter asks for the matching where the sum of the weights of the edges has a maximal value. There are two features that distinguish the SRBRA problem from maximum weighted bipartite matching which could be solved with Kuhn-Munkres algorithm [6,9] in polynomial time. Firstly, since each program has a fixed broadcasting time, a transmission device can be allocated to different programs if there is no clash in the broadcasting time. Hence the solution to the SRBRA problem is not a matching. Secondly, there are conflicts in the transmission resources, which means some edges cannot be selected simultaneously.

2.3 NP-hardness of SRBRA

We now prove the NP-hardness of the SRBRA problem.

Proposition 1. *The shortwave radio broadcast resource allocation problem is NP-hard.*

Proof. Consider the decision version of the SRBRA problem, which asks if it is possible to arrange m radio programs on n transmission devices so that the first two requirements in Definition 1 are satisfied. We can prove that this decision problem is NP-complete via reduction from the independent set problem. In graph theory, an independent set is a set of vertices in a graph, no two of which are adjacent. The problem of finding an independent set of certain size is a classical NP-complete problem in computer science. The general form of an independent set problem is: Given a graph G with n vertices, does there exist an independent set of size m ($m < n$)? We can construct a decision problem of SRBRA from the independent set problem by the following steps:

1. For each vertex v_j, create a transmission device D_j. If there is an edge connecting v_j and v_k in G, add a constraint $conflict(D_j, D_k)$.
2. Create m radio programs of identical broadcasting time. (The target areas can be arbitrary.)
3. Set the broadcasting effect of each allocation $< P_i, D_j >$ to be admissible, which means each program can be transmitted with all of the devices.

Apparently, the above procedure is a polynomial-time reduction. Suppose the SRBRA problem has a solution $\mathcal{S} = \{< P_i, D_j >\}$. The set of vertices corresponding to the devices that are allocated, denoted by $\mathcal{V} = \{v_j | \exists P_i < P_i, D_j >\in \mathcal{S}\}$, is a solution to the original independent set problem. This can be easily proved by reduction by contradiction. Firstly, the cardinality of $\mathcal{V}$ is m, otherwise, there must be at least one device allocated to two programs. Since the programs have the same broadcasting time, such allocations contradict the second requirement in Definition 1. Secondly, assume there are two adjacent vertices in $\mathcal{V}$, namely v_j and v_k. According to the construction, D_j and D_k are conflicting devices, and are used to transmit programs with the same broadcasting time. The second requirement in Definition 1 is violated too. Hence $\mathcal{V}$ is an independent set of size m. Conversely, if the independent set problem has a solution, the SRBRA problem is feasible, too.

Now that we have proved the NP-completeness of the decision version of the SRBRA problem, it easily follows that the problem itself is NP-hard.   □

3   Pseudo-Boolean Formulation

The above problem can be naturally formulated as a Pseudo Boolean Optimization Problem. In its broadest sense, a Pseudo Boolean function is a function that maps Boolean values to a real number. However, in this context, we only need linear functions with integer coefficients to express the objective and the constraints. A linear Pseudo-Boolean Optimization Problem can be formally defined as:

$$Maximize \qquad \Sigma_j c_j x_j$$

$$Subject\ to \quad \bigwedge_i \Sigma_j a_{ij} x_j \leq b_i$$

where $x_j \in \{0, 1\}$ is a Boolean variable.

3.1   Encoding

For each admissible allocation, say $< P_i, D_j >$, we introduce a Boolean variable $A_{i,j}$ to indicate whether the allocation is adopted. In other words, $A_{i,j}$ is `true` if and only if P_i is transmitted with D_j.

Recall that our goal is to maximize the total number of qualified sites in the target areas of all programs. The objective function is as follows:

$$Maximize \quad \Sigma_i \Sigma_j N_{<P_i, D_j>} \times A_{i,j}$$

There are three types of linear Pseudo-Boolean constraints:

1. It is required that each program is transmitted with exactly one device. For each program P_i $(1 \leq i \leq m)$, we add the following Pseudo-Boolean constraint:

$$\Sigma_{j=1}^n A_{i,j} = 1 \tag{1}$$

2. If two programs P_i and P_k clash in the broadcasting time, then they cannot be transmitted with the same device. So for each pair of such programs we add the following constraints:

$$\bigwedge_{1 \leq j \leq n} A_{i,j} + A_{k,j} \leq 1 \tag{2}$$

3. We also have to prevent conflicting devices from being used simultaneously. Suppose P_i and P_k are two programs clashing in broadcasting time. A natural way to encode this restriction is as follows:

$$\bigwedge_{conflict(D_u, D_v)} A_{i,u} + A_{k,v} \leq 1 \wedge A_{i,v} + A_{k,u} \leq 1 \tag{3}$$

However, in our implementation, we find the following constraints more effective than Formula (3).

$$\bigwedge_{conflict(D_u, D_v)} A_{i,u} + A_{i,v} + A_{k,u} + A_{k,v} \leq 1 \tag{4}$$

It states that if D_u and D_v are conflicting devices, then only one of them can transmit one of the programs. Note that Formula (4) implies some constraints in Formula (2). But this kind of redundancy in encoding proved to be beneficial.

4   The Local Search Algorithm

The Greedy Randomized Adaptive Search Procedure (also known as GRASP) [3,12] is a metaheuristic algorithm commonly applied to combinatorial optimization problems. Generally, it consists of iterations over two phases: the greedy randomized construction of an initial solution and subsequent iterative improvements of the solution. Normally a solution to a combinatorial optimization problem is composed of many elements. In the construction phase, the solution is built by iteratively adding elements. Each element is randomly selected from a list of elements ranked by some greedy function according to the quality of the solution they will achieve. The list is called a restricted candidate list (RCL). During the improvement phase, the algorithm tries to improve the constructed solution by searching in its neighborhood. The two-phase process is executed repeatedly.

Our local search algorithm employs the GRASP metaheuristic. We first introduce several basic sub-procedures before describing the whole algorithm, which are devised taking into consideration the structure of the SRBRA problem.

4.1   Consistency Checking

The second requirement in Definition 1 imposes an important set of constraints. We should guarantee that during search the solution $\mathcal{S}$, partial or complete, always satisfies these constraints. Consistency checking is frequently invoked in our local search algorithm and its efficiency is critical to the success of the algorithm. If the consistency checking process is time-consuming, the local search algorithm can only visit a small portion of the whole search space, thus is likely to miss solutions with high quality. In the SRBRA problem, there are so many constraints, thus checking efficiently if any constraint is violated is nontrivial.

We devise the sub-procedure $\texttt{Consistent}(< P_i, D_j >, \mathcal{S})$. It checks if the allocation $< P_i, D_j >$ is consistent with the current solution $\mathcal{S}$, as shown in Algorithm 1. Since each program is assigned only one device, the number of allocations in the current solution $\mathcal{S}$ is no more than the number of programs. Besides, in our implementation we use a two-dimensional matrix to store the conflicting relationship of devices, so $conflict(D_j, D_r)$ can be decided in constant time. The overall time complexity of $\texttt{Consistent}$ is linear in the number of programs. Since the number of programs is quite small compared with the number of devices, the sub-procedure $\texttt{Consistent}$ is very efficient.

Algorithm 1. Consistent($< P_i, D_j >, \mathcal{S}$)

1: **for** each $< P_k, D_r > \in \mathcal{S}$ **do**
2:   **if** $(conflict(D_j, D_r) \vee D_j = D_r) \wedge (t_i < t'_k \wedge t_k < t'_i)$ **then**
3:     return **false**;
4:   **end if**
5: **end for**
6: return **true**;

4.2   Greedy Randomized Construction

Algorithm 2 describes the greedy randomized construction procedure
Construct(). At first, the solution $\mathcal{S}$ is initialized as an empty set. In each iter-
ation, it randomly chooses a program P_i which has not been assigned a device.
All devices are then examined for eligibility to transmit the program. There
are two conditions for a device D_j to be eligible: Firstly, $< P_i, D_j >$ should be
admissible; Secondly, the allocation $< P_i, D_j >$ should be consistent with the
allocations already selected in the partial solution $\mathcal{S}$. The set of eligible devices
is denoted by $\mathcal{C}$. We simply adopt the number of qualified sites as the greedy
function. Therefore, the devices with the top 10 % highest numbers of qualified
sites are selected into the set $\mathcal{C}^*$. Then P_i is assigned a device randomly chosen
from $\mathcal{C}^*$.

Algorithm 2. Construct()

1: Solution $\mathcal{S} = \phi$;
2: **repeat**
3:     randomly choose an unassigned program $P_i \in \mathcal{P}$;
4:     $\mathcal{C} = \phi$;
5:     **for** each device $D_j \in \mathcal{D}$ **do**
6:        **if** $< P_i, D_j >$ is admissible and Consistent($< P_i, D_j >$,$\mathcal{S}$) **then**
7:           $\mathcal{C} = \mathcal{C} \cup \{D_j\}$;
8:        **end if**
9:     **end for**
10:    $\mathcal{C}^* = \{D_j | N_{<P_i,D_j>} \text{ is within the top } 10\% \text{ in } \mathcal{C}\}$;
11:    randomly choose a device D_r from $\mathcal{C}^*$;
12:    $\mathcal{S} = \mathcal{S} \cup \{< P_i, D_r >\}$;
13: **until** $\mathcal{P}$ is traversed.
14: return $\mathcal{S}$;

4.3   Operations in the Improvement Phase

We propose two operations that can locally improve a solution $\mathcal{S}$, Swap($\mathcal{S}$) and
Substitute($\mathcal{S}$).

The Swap($\mathcal{S}$) operation illustrated in Algorithm 3 improves the solution $\mathcal{S}$ by
selecting a pair of allocations in $\mathcal{S}$ and exchanging the corresponding devices. The
allocation pair is greedily selected according to a rank function that evaluates
the benefit of the exchange. For two allocations $< P_i, D_j >$ and $< P_k, D_r >$
in $\mathcal{S}$, swapping the devices would generate two new allocations $< P_i, D_r >$ and
$< P_k, D_j >$. The new allocations should be both admissible and consistent with
other allocations in $\mathcal{S}$. The resulting benefit is defined as the increment in the
number of qualified sites, or formally:

$$score(\{< P_i, D_j >, < P_k, D_r >\}) = N_{<P_i,D_r>} - N_{<P_i,D_j>} + N_{<P_k,D_j>} - N_{<P_k,D_r>}$$

Finally, the pair of allocations with the highest score is swapped.

Algorithm 3. Swap($\mathcal{S}$)

1: $maxscore = 0$;
2: $bestpair = newpair = \phi$;
3: **for** any two allocations $< P_i, D_j >$ and $< P_k, D_r >$ in $\mathcal{S}$ **do**
4:    **if** $< P_i, D_r >$ and $< P_k, D_j >$ are both admissible **then**
5:       $pair = \{< P_i, D_j >, < P_k, D_r >\}$;
6:       **if** Consistent($< P_i, D_r >$, $\mathcal{S} \setminus pair$) and Consistent($< P_k, D_j >$, $\mathcal{S} \setminus pair$) **then**
7:          $score = N_{<P_i,D_r>} - N_{<P_i,D_j>} + N_{<P_k,D_j>} - N_{<P_k,D_r>}$;
8:          **if** $score > maxscore$ **then**
9:             $maxscore = score$;
10:            $bestpair = pair$;
11:            $newpair = \{< P_i, D_r >, < P_k, D_j >\}$;
12:          **end if**
13:       **end if**
14:    **end if**
15: **end for**
16: $\mathcal{S} = (\mathcal{S} \setminus bestpair) \cup newpair$;
17: return $\mathcal{S}$;

Algorithm 4. Substitute($\mathcal{S}$)

1: $maxscore = 0$;
2: $alloc1 = null, alloc2 = null$;
3: **for** Each allocation $< P_i, D_j > \in \mathcal{S}$ **do**
4:    **for** Each device $D_r \in (\mathcal{D} \setminus \{D_u | \exists P_k(< P_k, D_u > \in \mathcal{S})\})$ **do**
5:       **if** $< P_i, D_r >$ is admissible and Consistent($< P_i, D_r >$, $\mathcal{S} \setminus \{< P_i, D_j >\}$) **then**
6:          $score = N_{<P_i,D_r>} - N_{<P_i,D_j>}$;
7:          **if** $score > maxscore$ **then**
8:            $maxscore = score$;
9:            $alloc1 = < P_i, D_j >$;
10:            $alloc2 = < P_i, D_r >$;
11:          **end if**
12:       **end if**
13:    **end for**
14: **end for**
15: $\mathcal{S} = (\mathcal{S} \setminus \{alloc1\}) \cup \{alloc2\}$
16: return $\mathcal{S}$;

The Swap operation is a minor adjustment within the solution. The search procedure is likely to get stuck in the local optimum with respect to the neighborhood structure merely specified by Swap. Therefore, we introduce another operation Substitute($\mathcal{S}$) which is described in Algorithm 4. The basic idea is to select an allocation $< P_i, D_j >$ in the current solution $\mathcal{S}$ and replace D_j with an idle device D_r. The selection is based on the greedy function evaluating the benefit of the substitution, which is defined as:

$$score(< P_i, D_j >, D_r) = N_{<P_i, D_r>} - N_{<P_i, D_j>}$$

The new allocation $< P_i, D_r >$ must be admissible and consistent with other allocations in $\mathcal{S}$, too. Obviously, the `Substitute` operation changes the set of allocated devices, thus may improve the quality of the solution.

4.4   The Local Search Procedure

The framework of our local search algorithm is shown in Algorithm 5, as described below. During the search, $\mathcal{S}^*$ keeps the best solution found thus far. In the beginning, an initial solution $\mathcal{S}$ is constructed, and the counters *swaps* and *substitutes* are set to 0. Then the algorithm iterates over the greedy randomized construction phase `Construct` and two improvement operations `Swap` and `Substitute` until the cut-off time is reached. There are two control parameters, I and R. At each step, if the current solution $\mathcal{S}$ is better than the best solution $\mathcal{S}^*$, the algorithm updates $\mathcal{S}^*$ and resets both of the counters. Otherwise, the `Swap` operation is firstly applied to improve the objective value $N_{\mathcal{S}}$. If `Swap` is applied up to I times and there is no improvement over $\mathcal{S}^*$, the `Substitute` operation is applied to improve solution $\mathcal{S}$, and the counter *swaps* is cleared. If $\mathcal{S}^*$ is still not improved by performing the `Substitute` operation R times, the algorithm restarts to get out of local optimum.

5   Empirical Evaluation

In this section we present experimental results of the proposed approaches on the set of real size shortwave radio broadcasting instances provided by the Division of Radio Frequency Assignment of SAPPRFT. These instances are taken from 7061 radio transmission devices and 87 programs. All instances are available on the website[2]. We use the solver `clasp`[3] for PBO solving. Since the PBO formulation of the SRBRA problem is naturally a 0-1 integer linear programming (ILP) problem, we also employ `CPLEX` as the ILP solver in the experiments.

We carried out two sets of experiments. The first one aims to compare the local search (LS) algorithm with the PBO approach and ILP solving, while the second one is mainly to evaluate the performance of the LS algorithm on large scale instances. The time limit for the LS algorithm is 10 seconds and for `clasp` and `CPLEX` is 3600 seconds. The number of qualified sites (denoted by #QS) obtained by each method is listed. Because of the randomness of the LS algorithm, it is executed 10 times for each instance, and both the maximum and average results are listed. Since `clasp` and `CPLEX` are exact solvers, if they terminate within the time limit, an optimal solution is found, and the corresponding result is marked with a star(*). Otherwise, we report the time they take to reach the primal solution at timeout. The times are measured in seconds. All experiments are performed on a computer with 3.3 GHz and 4 GB RAM under windows 7.

---

[2] http://ai.nenu.edu.cn/yinmh/.
[3] http://potassco.sourceforge.net/.

Algorithm 5. The Local Search Algorithm

```
 1: Solution S*=S = Construct();
 2: swaps = 0, substitutes = 0;
 3: while elapsed time < cut-off time do
 4:    if N_{S*} < N_S then
 5:       S* = S;
 6:       swaps = 0;
 7:       substitutes = 0;
 8:    else
 9:       if (swaps < I) then
10:          swaps = swaps + 1;
11:          S=Swap(S);
12:       else
13:          if (substitutes < R) then
14:             substitutes = substitutes + 1;
15:             swaps = 0;
16:             S = Substitute(S);
17:          else
18:             S = Construct();
19:             swaps = 0;
20:             substitutes = 0;
21:          end if
22:       end if
23:    end if
24: end while
25: return S*;
```

Table 1 shows the results of comparing the LS algorithm with `clasp` and `CPLEX`. The number of devices is denoted by 'D" and the number of programs by "P". We can observe from Table 1 that `CPLEX` performs best. It finishes searching within one second for all instances, providing the optimal value. The LS algorithm produces a solution in less than one second for almost all instances, while `clasp` reaches the time limit for some instances. The LS algorithm can find the exactly optimal solution for many instances.

Table 2 shows the experimental results on large instances. `CPLEX` reports "out of memory" and does not provide any solution for all these instances, so we only list the results of `clasp` and the LS algorithm. The "total" is the total number of sites in the instance. The "rate" is cover rate, that is "#QS/total". The "UB" column gives the upper bound of #QS for each instance, which is obtained by simply selecting the best device for each program, ignoring the device conflicting constraints. The symbol '-' indicates that `clasp` fails to produce a solution at timeout. For those `clasp` does produce solutions, the results are unsatisfactory. By contrast, LS can finish all the instances within 10 seconds, with high cover rate meeting the need. The instance (D = 7061, P = 87) contains all usable official radio transmission devices and programs, and takes only 9.62 s

with 81.1 % average cover rate. That means the LS algorithm can allocate all shortwave radio broadcast resource in China in a short time with high quality.

We also studied how the solution quality evolves as time passes for LS and `clasp` on two representative instances, as demonstrated in Fig. 3. The left picture indicates that for the instance with 400 devices and 20 programs, LS converges to the optimal solution much faster than `clasp` does. The picture on the right shows that the LS algorithm quickly reaches a good solution on the largest instance.

6   Related Works

The SRBRA problem is an extended resource allocation problem, which is a well-known problem widely studied in the literature. Numerous algorithms for variation of resource allocation problem are proposed in the decades. For example, for general resource allocation problem, [8] gives a hybrid search algorithm which combines genetic algorithm (GA) and ant colony optimization (ACO). Meanwhile special cases of resource allocation problem are studied widely. For example, [11] gives an algorithm for multi-vehicle systems with nonholonomic constraints. In this problem the vehicles satisfy a nonholonomic constraint, and the algorithm employs ideas from the traveling salesman problem and the path planning literature. [13] presents a proportional share resource allocation algorithm for real-time, time-shared systems.

Table 1. Comparison of `clasp`, `CPLEX` and LS

D	P	clasp		CPLEX		LS	
		#QS	time	#QS	time	max(avg) #QS	time
50	2	94*	<0.01	94*	<0.01	94(94)	<0.01
50	5	103*	0.171	103*	0.19	103(103)	<0.01
100	2	94*	0.016	94*	<0.01	94(94)	<0.01
100	5	103*	0.171	103*	0.14	103(103)	<0.01
200	2	121*	0.078	121*	0.14	121(121)	<0.01
200	5	238	7.269	238*	0.16	238(238)	<0.01
200	10	762	273.995	762*	0.27	762(762)	0.014
400	2	187*	0.577	187*	0.13	187(187)	<0.01
400	5	438	25.569	438*	0.09	438(438)	0.015
400	10	965	473.975	965*	0.19	965(965)	0.047
400	20	1228	3200.502	1232*	0.28	1232(1231.9)	3.411
800	2	212*	6.209	212*	0.16	212(212)	0.026
800	5	501	1589.128	501*	0.14	501(501)	0.127
2000	10	1305	1072.545	1555*	0.23	1555(1554.9)	2.667
3000	10	1393	1977.724	1562*	0.28	1562(1562)	1.284
4000	10	1246	1578.536	1677*	0.69	1677(1677)	1.006

Table 2. Experiments on large instances

| D	P	clasp		LS				UB
		#QS	time	max(avg) #QS/total	max(avg) rate(%)	time	
2000	20	1830	2169.083	2183(2179.8)/2677	81.5(81.4)	4.180	2251
2000	30	710	3091.099	2464(2455.5)/3081	80.0(79.7)	3.260	2597
2000	40	1048	3177.101	3526(3484.2)/4618	76.4(75.4)	3.953	3858
2000	50	1020	1670.685	4180(4116.3)/5655	73.9(72.8)	3.510	4739
2000	60	847	991.647	4872(4806)/6687	72.9(71.9)	4.959	5619
2000	70	1182	2026.448	5707(5611.3)/8055	70.9(69.7)	6.137	6826
2000	87	-	-	6358(6234.2)/9404	67.6(66.3)	8.618	7981
3000	20	649	2286.309	2197(2192.4)/2677	82.1(81.9)	1.639	2251
3000	30	140	490.964	2502(2493.7)/3081	81.2(80.9)	2.528	2599
3000	40	540	377.552	3634(3591.1)/4618	78.7(77.8)	4.308	3860
3000	50	1014	1345.908	4306(4265.1)/5655	76.1(75.4)	5.971	4741
3000	60	1117	2112.735	5040(4996.8)/6687	75.4(74.7)	8.581	5623
3000	70	-	-	5969(5918)/8055	74.1(73.5)	9.934	6830
3000	87	-	-	6666(6609.7)/9404	70.9(70.3)	9.907	7990
4000	20	250	78.608	2320(2318)/2677	86.7(86.6)	2.716	2370
4000	30	115	853.837	2633(2626)/3081	85.5(85.2)	3.227	2753
4000	40	390	2998.060	3822(3800.8)/4618	82.8(82.3)	5.806	4092
4000	50	1060	3004.924	4577(4541.7)/5655	80.9(80.3)	8.500	5063
4000	60	-	-	5387(5329.4)/6687	80.6(79.7)	9.884	6040
4000	70	-	-	6284(6249.9)/8055	78.0(77.6)	9.806	7315
4000	87	-	-	7029(6986.5)/9404	74.7(74.3)	9.824	8570
5000	10	1182	2765.580	1682(1682)/1954	86.1(86.1)	1.312	1695
5000	20	320	2625.094	2338(2334.7)/2677	87.3(87.2)	2.493	2380
5000	30	514	76.035	2679(2660.7)/3081	87.0(86.4)	3.993	2763
5000	40	530	2324.841	3893(3877.9)/4618	84.3(84.0)	7.029	4103
5000	50	927	2892.373	4708(4681.5)/5655	83.3(82.8)	9.845	5077
5000	60	-	-	5496(5416.3)/6687	82.2(81.0)	9.912	6054
5000	70	-	-	6373(6318)/8055	79.1(78.4)	9.795	7335
5000	87	-	-	7186(7142.9)/9404	76.4(76.0)	9.712	8594
6000	10	719	148.395	1772(1772)/1954	90.7(90.7)	1.555	1783
6000	20	391	66.634	2456(2453.6)/2677	91.7(91.7)	2.805	2473
6000	30	411	199.213	2810(2804.3)/3081	91.2(91.0)	4.868	2861
6000	40	874	2329.879	4164(4137)/4618	90.2(89.6)	8.495	4307
6000	50	-	-	5030(4990.4)/5655	88.9(88.2)	9.803	5293
6000	60	-	-	5856(5788.9)/6687	87.6(86.6)	9.826	6288
6000	70	-	-	6878(6797.5)/8055	85.4(84.4)	9.842	7592
6000	87	-	-	7854(7667.2)/9404	83.5(81.5)	9.687	8875
7061	10	696	2734.533	1774(1774)/1954	90.8(90.8)	2.269	1785
7061	20	337	203.089	2458(2456.3)/2677	91.8(91.8)	3.358	2475
7061	30	249	2665.458	2812(2802.8)/3081	91.3(91.0)	5.796	2863
7061	40	629	1492.673	4192(4183.2)/4618	90.8(90.6)	9.778	4311
7061	50	-	-	5068(5045.9)/5655	89.6(89.2)	9.828	5297
7061	60	-	-	5883(5805.7)/6687	88.0(86.8)	9.909	6301
7061	70	-	-	6860(6800.8)/8055	85.2(84.4)	9.779	7605
7061	87	-	-	7739(7629.6)/9404	82.3(81.1)	9.620	8888
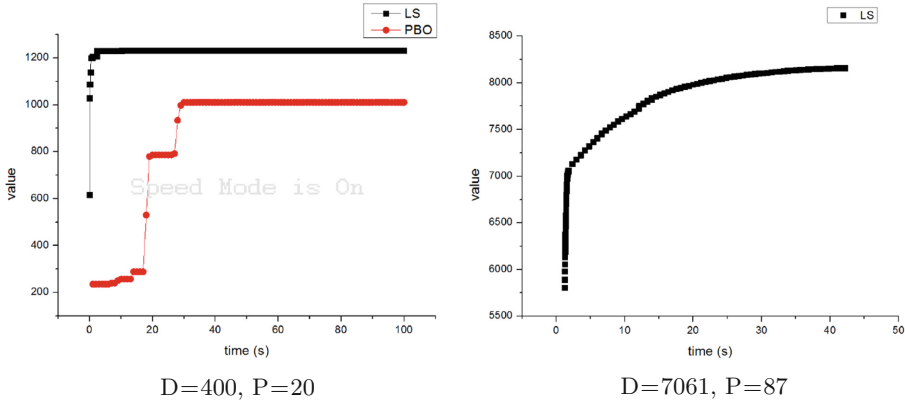
Fig. 3. Solution vs Time

For the case of radio, some effective algorithms are proposed to tackle the allocation and scheduling problem. [14] proposes a scheduling algorithm based on mean field annealing (MFA) neural networks for an optimal broadcast scheduling in packet radio networks. [10] also proposes a scheduling algorithm for multi-hop radio networks. [5] presents a radio resource allocation algorithm for Relay-Aided Cellular OFDMA System.

Some methods for the allocation and schedule problem for radio are based on CSP and Tabu Search. For example, [15] presents an algorithm for solving the frequency assignment problem (FAP) in cellular mobile systems, which uses CSP techniques. The algorithm represents a cell as a variable that has a very large domain, and determines a variable value step by step. [3] presents a tabu search algorithm for the FAP in mobile radio networks. [4] presents a modeling approach of the interference constraints and a probabilistic taboo search algorithm to solve the FAP in broadcasting. For more about FAPs, one can visit the website [1].

7   Conclusion

Shortwave radio broadcasting is still in heavy use in countries like China. In this paper, we studied the shortwave radio broadcast resource allocation problem (SRBRA), which is a kind of optimization problem with complex constraints. The SRBRA problem is derived from a real project in which we are cooperating with the Division of Radio Frequency Assignment of SAPPRFT. We have described the problem formally, and proved that it is an NP-hard problem. We proposed two complementary methods to solve SRBRA. One is to formulate it as a Pseudo-Boolean optimization (PBO) problem; and the other is a local search algorithm with quick consistency checking mechanism and the metaheuristic of Greedy Randomized Adaptive Search Procedure (GRASP). We have implemented both methods and evaluated them with real data from the Division of

Radio Frequency Assignment of SAPPRFT. It turns out that, we can find an allocation plan with good broadcasting effect quite efficiently.

References

1. Frequency Assignment Problem. http://fap.zib.de/
2. Hand, G.: VOACAP, ICEPAC and REC-533 propagation prediction programs for windows. *NTI/ITS*
3. Hao, J.-K., Dorne, R., Galinier, P.: Tabu search for frequency assignment in mobile radio networks. J. Heuristics **4**(1), 47–62 (1998)
4. Idoumghar, L., Debreux, P.: New modeling approach to the frequency assignment problem in broadcasting. IEEE Trans. Broadcast. **48**(4), 293–298 (2002)
5. Kaneko, M., Popovski, P.: Radio resource allocation algorithm for relay-aided cellular ofdma system. In: Proceedings of IEEE International Conference on Communications, pp. 4831–4836, June 2007
6. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Res. Logistics Q. **2**(1–2), 83–97 (1955)
7. Lane, G.: Signal-to-Noise Predictions Using VOACAP-A Users Guide. Rockwell Collins Inc., USA (2001)
8. Lee, Z.-J., Lee, C.-Y.: A hybrid search algorithm with heuristics for resource allocation problem. Inf. Sci. **173**(1–3), 155–167 (2005)
9. Munkres, J.: Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**, 32–38 (1957)
10. Ramanathan, S., Lloyd, E.L.: Scheduling algorithms for multihop radio networks. IEEE/ACM Trans. Netw. **1**(2), 166–177 (1993)
11. Rathinam, S., Sengupta, R., Darbha, S.: A resource allocation algorithm for multivehicle systems with nonholonomic constraints. IEEE Trans. Autom. Sci. Eng. **4**(1), 98–104 (2007)
12. Resende, M.G.C., Ribeiro, C.C.: Greedy randomized adaptive search procedures: advances, hybridizations, and applications. In: Handbook of Metaheuristics, pp. 283–319. Springer, Boston (2010)
13. Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, S.K., Gehrke, J.E., Plaxton, C.G.: A proportional share resource allocation algorithm for real-time, time-shared systems. In: Proceedings of IEEE Real-Time Systems Symposium, pp. 288–299, December 1996
14. Wang, G., Ansari, N.: Optimal broadcast scheduling in packet radio networks using mean field annealing. IEEE J. Sel. Areas Commun. **15**(2), 250–260 (1997)
15. Yokoo, M., Hirayama, K.: Frequency assignment for cellular mobile systems using constraint satisfaction techniques. In: Proceedings of IEEE Vehicular Technology Conference, vol. 2, pp. 888–894 (2000)

Availability Optimization in Cloud-Based In-Memory Data Grids

Samir Sebbah[✉], Claire Bagley, Mike Colena, and Serdar Kadioglu

Oracle Corporation, Burlington, MA, USA
{samir.sebbah,claire.bagley,mike.colena,serdar.kadioglu}@oracle.com

Abstract. This paper presents a Constraint Programming (CP)-based application for dynamic cache distribution in Oracle Coherence In-Memory Data Grid (IMDG). A re-sizable decomposition method using CP is developed to ensure high availability through incremental optimization of load distribution and data replication. The application highlights the flexibility and efficiency that the CP technology offers for *(1)* concisely capturing the multiple dynamic aspects and complex constraints of the Oracle Coherence IMDG cache distribution problem; and *(2)* solving large-scale problem instances in a dynamic cloud environment. Extensive computational results are presented to assess the scalability and efficiency of the proposed solution.

Keywords: Constraint programming · Re-sizable decomposition · Oracle Coherence · Dynamic Cache Distribution · Availability · Resiliency

1 Introduction

The advent of cloud computing has redefined the ways many businesses are operating. The ability to rapidly build and deploy scalable applications/services and obtain/release computing resources in a cost-effective and device-independent way have driven many companies to switch from on-premise IT solutions to the cloud. In-Memory Data Grid (IMDG) is a one of the big-data applications that has been experiencing a big shift from on-premise to the cloud. A cloud-based IMDG is a distributed data structure store that keeps data in memory for providing low access latency and high availability to mission critical and big-data applications. By loading Tera-bytes of data into a distributed memory store, IMDGs can meet the stringent big-data processing requirements in terms of scalability, resiliency, efficiency, and availability [2]. However, multiple challenges need to be constantly monitored in order for an IDMG to function correctly and provide the expected Quality-of-Service (QoS) guarantees [1,5]. In IMDGs, maintaining data integrity with 100 % transactional data handling is extremely important for ensuring system data consistency. Nowadays, many IMDGs, on premise or in the cloud, support functionalities that provide elastic provisioning to meet fluctuating and variable unexpected loads, asynchronous data replication, smart data routing, load balancing, and scalability.

Among all the QoS metrics [5], availability is the metric to optimize for the success of any business in the cloud. The availability of a cloud system is usually expressed as the percentage of up-time in a given year, e.g., 99.999 %, ~5.26 minutes of down-time per year. Service unavailability, also expressed as the percentage of down-time per year, is due to several reasons nominally attributed to software, hardware, or network infrastructure failures. Service unavailability in the cloud may also be due to congestion of the cloud infrastructure caused by unbalanced workload distribution and/or inefficient use of resources. In the cloud community, notable efforts are centered around the area of availability provisioning [1]. To achieve any availability level, the cloud system needs to be resilient against failures. Existing mechanisms to mitigate and anticipate the impacts of failures in computing and networking have widely been re-used in the cloud. Examples include dynamic resource allocation, resiliency, and load balancing [1,6]. Many QoS centered problems in the cloud are combinatorially NP-hard [3,13]. Given the growing complexity and size of the cloud model, finding solutions to those combinatorial problems is becoming tremendously hard in practice. Furthermore, the evolving nature of the cloud [9] poses other challenges related to how quickly and economically a solution that addresses current needs can be extended and maintained to respond to future requirements. Therefore, optimization solutions to embed QoS provisioning and resiliency are expected to go beyond the cost-driven model and address multiple other business and operation aspects of the cloud.

In this paper, we present a Constraint Programming (CP)-based re-sizable decomposition method for solving the Dynamic Cache Distribution Problem (DCDP) in Oracle Coherence cloud-based IMDG. Oracle Coherence is the industry leading IMDG solution that enables organizations to predictably scale mission-critical applications [11]. The DCDP is formulated as a variant of the bin-packing problem and solved using a re-sizable decomposition that uses CP to incrementally optimize the availability of the Oracle Coherence IMDG system.

The paper is organized as follow. Section 2 presents the DCDP and the architecture of the Oracle Coherence IMDG system. Section 3 details the solution approach we developed for solving the DCDP. Section 4 presents computational results assessing the performance of the proposed approach. Section 6 concludes the paper.

2   The DCDP in the Oracle Coherence IMDG

2.1   Architecture

Figure 1 presents a model of the cloud-based Oracle Coherence IDMG system. The cloud physical infrastructure is composed of an interconnection of data centers that provide processing capacity, storage, and network services for the IMDG system. Server (or platform) virtualization is the technique that allows a big computer system to be partitioned into multiple isolated execution environments similar to a single physical computer, also called Virtual Machines
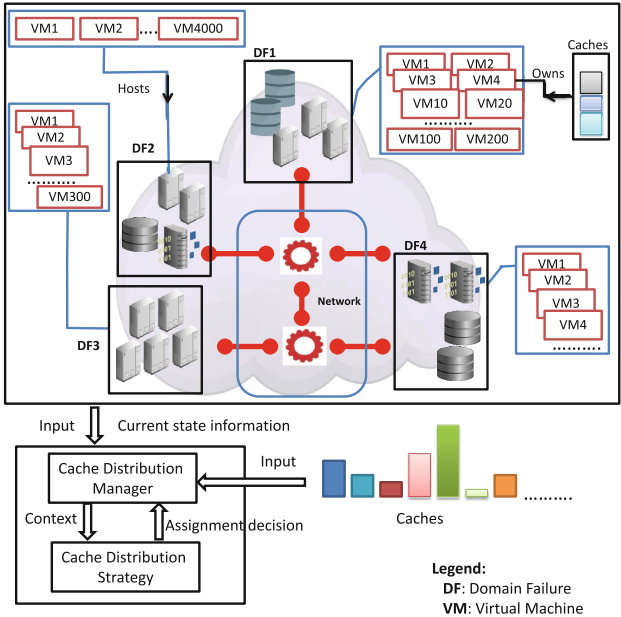
Fig. 1. A Model of the Oracle Coherence IMDG System.

(VM). Each VM can be configured in an independent way to have its own operating system, memory capacity, and network parameters [14]. In Fig. 1, physical machines/data centers that *share the same risk of failure* are grouped together into a logical entity called a Domain Failure (DF).

The Oracle Coherence IMDG system is a distributed data structure repository that resides entirely in memory [11]. To support the increasing demands for big-data applications, Oracle Coherence is designed to support hundreds of thousands of operations per second by providing continuous data availability in memory. In Oracle Coherence, the data of users is divided into multiple dynamic caches and stored in the distributed in-memory repository. When deployed in the cloud, the Oracle Coherence IMDG is granted access to a set of resources that it can rapidly provision and release to meet any demand fluctuation [11]. The IMDG model in Fig. 1 uses the memory of the VMs to store the dynamic caches of the users. The current state information of all the resources (locations/sizes of the caches, load of the VMs, state of the machines, e.g., up/faulty) and the demands coming from the users are given as inputs.

The IMDG decision component is composed of two modules: the Cache Distribution Manager (CDM) and the Cache Distribution Strategy (CDS) [11]. The CDM maintains the state information of the IMDG system, assigns new caches, and re-assign (migrate) pre-assigned caches. In the assignment process, the CDM takes assignment recommendations from the CDS which instantiates and solves cache distribution decision problems based on current state information.

The CDM may query the CDS for assignment decisions at regular time intervals or after any of the following state information change in the IDMG system: (1) load distribution unbalance, (2) resource upgrade (3) resource removal - gracefully or accidentally (due to failures). Our CP-based cache distribution strategy for the DCDP will function as the CDS module as shown in Fig. 1.

2.2   Requirements

The availability of any IMDG system can be optimized by a resource allocation strategy that embeds QoS provisioning in its allocation activities. In the following we discuss some of the requirements for achieving optimized high-availability in Oracle Coherence IDMG system.

– Resiliency: guarantee 100 % survivability against any k simultaneous machine failures, i.e., the IMDG system will continue to function even when k machines are in failure state and cannot provide any service. In case of any machine failures, the system will switch over to backup machine(s) and continue to provide the same service (with the same quality) as before the failure happened. To achieve a k resiliency level [2], k replicas (backup caches) of each primary cache are created and assigned resources in different DFs.
– Load balancing: guarantee an even load distribution across the candidate VMs. The load balancing metric is used for access latency prediction and optimization [1].
– Fast provisioning and scalability: time consuming provisioning efforts add to the unavailability of the IMDG system and negatively affect the scalability of the system. Therefore, fast provisioning decisions are wanted for improved scalability. Furthermore, given that resources are dynamically added to and removed from the system, the IMDG needs to quickly update the state information and integrate them in the provisioning process.
– Provisioning pattern: incremental batch provisioning is required to minimize the duration of the cache distribution phase. During the cache distribution phase parts of the data become unavailable. To minimize the impact on the customers, the provisioning time[1] needs to be minimized, i.e., the CDS needs to quickly return suggestions of cache distribution and the CDM needs to implement them as soon as they arrive (minimum time gap). The CDM should never be in a lengthy idle state waiting for suggestions to come from the CDS.

In the next section, we present a re-sizable decomposition method that addresses the requirements above.

3   The CP-Based Solution Approach

The Coherence DCDP is a particular resource allocation problem in which stringent technical/business requirements are mandated within the Coherence software architecture. Conceptually, the DCDP can be modeled as a variant of the

---

[1] The time that spans from the start of decision making to the end of decision implementation.

widely used *bin-packing* approach for resource allocation in cloud computing [4,7,14]. Formally, the DCDP consists in finding a packing/re-packing pattern of dynamic data caches (items) of different sizes into a finite number of capacitated VMs (bins) in a way that maximize the items availability. The availability of items is enforced by resiliency constraints to forbid collocations of copies of the same item (copies of the same item cannot be on the same VM), and load balancing constraints to minimize data access latency. In addition, given that the data caches are continually and independently growing over time, a (re-)packing of the caches will need to be performed in a way that minimizes load migration across the VMs.

The NP-hardness of the bin-packing problem motivated many researchers to use heuristics especially for dynamic resource allocation in the cloud [13]. CP has been widely used for its expressiveness that fits well with the complex and evolving nature of the cloud model. The expressiveness of CP has been complemented with several studies that showed its effectiveness at solving variants of the bin packing problem [10,12].

In this section, we present our solution approach for solving the Oracle Coherence DCDP. The solution is composed of a Round-Robin (R-R) heuristic and a CP-based re-sizable decomposition.

3.1   Motivation

As mentioned in the requirements section, in order to optimize data availability perfect synchronization between decision making/execution is needed to minimize the provisioning time. In Coherence IMDG system, batches of cache distribution recommendations are sent by the CDS to the CDM for implementation. To achieve good synchronization between the CDS and the CDM, the size of each decision batch has to be tailored to the current state, including work load, of the IMDG system. Figure 2 illustrates three scenarios of synchronization between the CDS and CDM.

(a) The CDS is slower than the CDM (no synchronization): CDS solves large decision models while the CDM is waiting for decisions to come.
(b) The CDS is faster than the CDM (no synchronization): CDS solves small decision models and overwhelms the CDM with decisions.
(c) CDS and CDM are synchronized (ideal scenario): CDS solves ideally sized decision models to achieve synchronization with the CDM. No wasted time due to decision queuing or idle state is recorded, and the provisioning time is the lowest over all the three scenarios.

To achieve a perfect synchronization between the CDS and CDM while solving large-scale instances of the problem, we developed a re-sizable decomposition method where the size of the cache distribution decision problem can be dynamically changed over time. In the next sections we present the different building blocks of the re-sizable decomposition.
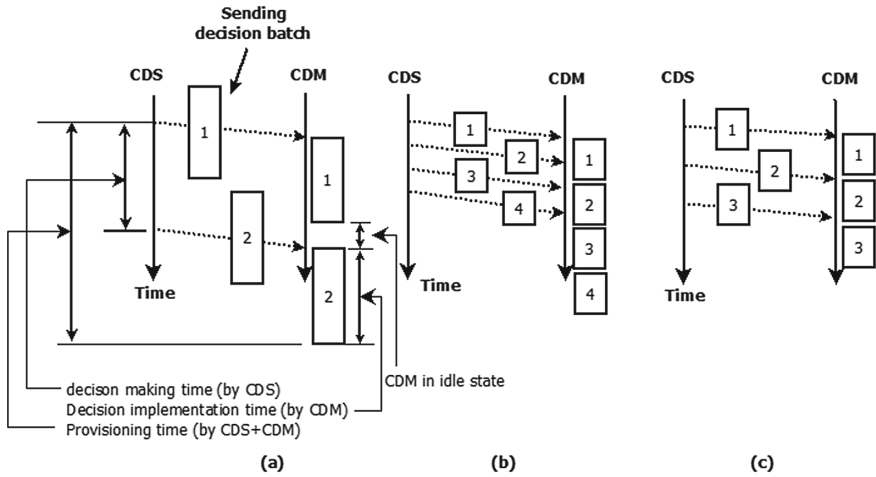
Fig. 2. Synchronization Between CDS and CDM.

3.2 The Replication Map

The DCDP involves assigning both primary caches and replicas (up to k replicas for each primary cache) to VMs. Provided that each primary cache and its replicas are all located in different DFs, each level of replication is a symmetrical problem of the primary cache distribution problem. In order to mirror the primary cache assignments in the replicas assignments, we build a *replication map* that defines the locations of all the replicas. Figure 3 shows an example of a replication map with replication paths indicating the locations of the k replicas of any primary cache (the arrows indicates the replication paths). Example, if a primary cache p_1 is assigned to DF4:VM1 then its first and second replicas will be along the replication path that starts at DF4:VM1, i.e., first replica will be on DF3:VM1 and second replica will be on DF1:VM1. To reach level k of resiliency, each path is required to visit k distinct DFs (no loops) in addition to the source of the primary cache (more details are provided in Sect. 3.6). The set of all replication paths defines a replication map and a routing table [3].

3.3 The Warm-Starting Heuristic

We use a Round-Robin (R-R) heuristic to quickly find a first solution that spreads the primary caches across the given VMs. The R-R heuristic alternates between forward and backward cache assignments, i.e., assign caches to $\{vm_1, vm_2, \ldots, vm_n\}$ then to $\{vm_n, vm_{n-1}, \ldots, vm_1\}$. The sizes of caches are not explicitly used in the process. For a more balanced cache assignment solution, the caches are sorted prior to the R-R assignment process.

By assigning caches to VMs, the R-R heuristics spreads almost equally the caches across the VMs, i.e., each VM gets almost equal number of caches.
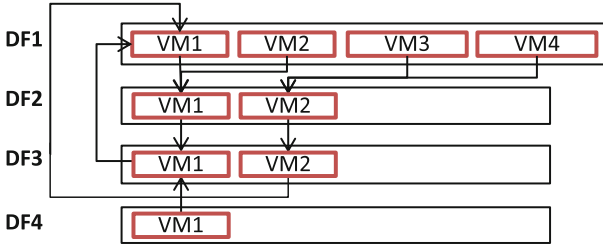
Fig. 3. An Example of a Replication Map with Resiliency Level $k = 2$.

From a combinatorial point of view, the R-R is creating almost equal-size decision models for the re-sizable decomposition to choose from.

3.4   The Re-sizable Decomposition

The following algorithm illustrates the re-sizable decomposition.

```
program The Re-Sizable Decomposition (decSize, deviationUpperBound)
   const
     MaxNbrIterations = 2* |VMs|;
   var
     NbrIterations: 0..MaxNbrIterations;
   begin
     NbrIterations := 0
     repeat
       NbrIterations := NbrIteration + 1;
       subProblem := DefineSubProblem (decSize);
       BuildSolveCPModel (subProblem);
       IF (LoadDeviationOfWholeInstance () <= deviationUpperBound)
         break;
       EndIf;
     until (NbrIterations == MaxNbrIterations)
   end.
```

The algorithm iterates through the following operations:

- The $DefineSubModel$ procedure defines the re-sizable part of the decomposition (illustrated in $DefineSubProblem$ algorithm below). Based on a decomposition size value $decSize$, the method selects $decSize$ of the most loaded VMs and the same number of least loaded VMs. The model grows in size with the decomposition size value. That is, $decSize = |VNs|/2$ is the whole model of the IMDG instance.
- Build and solve the CP sub-model ($BuildSolveCPModel$). The CP method is described in Sect. 3.5.

– Check the load deviation of the instance ($LoadDeviationOfWholeInstance$). The solution of each CP sub-model improves the load distribution across the VMs of the sub problem and consequently the load distribution of the whole IMDG instance. The decomposition stops when the load deviation of the IMDG instance is below the given threshold value ($deviationUpperBound$).

```
program DefineSubProblem (decSize)
  {Required decSize in {1, 2,..., |VMs|/2}};
   begin
      candidateVMs := getMostLoadedVMs (decomposition size)
                   + getLeastLoadedVMs (decomposition size)
      return candidateVMs;
   end.
```

This decomposition approach is conceptually similar to Column Generation where the sub-models are built on-the-fly based on the dual values associated with the constraints of the main model. At each iteration of the decomposition, a sub-model is built and solved to improve a global objective of load distribution. The sub-model of each iteration is built with variables associated with the most and least loaded VMs and their owned caches. Given that the load of the VMs change as the decomposition iterates, the model of the next iteration is always different from the one of the previous iteration.

3.5   The CP Model

The CP model is built with the set of VMs and their owned caches of the input problem.

The Variables

– Load (for each $vm \in VMs$)

$$Load_{vm} \in [minLoad \ldots maxLoad]$$

captures the load of each $vm \in VMs$. $minLoad$ and $maxLoad$ are the loads of the least loaded VM and of the most loaded VM, respectively.
– Assignment (for each cache $c \in Caches$)

$$Assign_c \in \{VMs\}$$

captures the assignment of each of the caches $c \in Caches$. The domain of the assignments is the set of VMs of the input problem.

The Constraints

– Bin-packing operator [12]

$$binPacking(Assign_c[], Size_c[], Load_{vm}[])$$

where $Size_c[]$ are the sizes of the caches. This constraint defines the assignment of the caches based on the *Load* array.
– Deviation operator [8,12]

$$deviation(Load_{vm}[])$$

This constraint, which measures the load deviation of the VMs, enforces cache load balancing.

Variable/Value Orderings
The following variable/value orderings are used to discriminate among solutions.

– Assign caches in decreasing order of size,
– Migrate caches (re-assign) only to improve load deviation.

3.6   The Replication Problem

The replication map, as introduced in Sect. 3.2, is the structure that indicates the locations of the k backup caches of each primary cache. The map example given in Fig. 3 can survive any $k = 2$ simultaneous DFs. To obtain the locations of the 2 backup caches of any primary cache, it is required to follow the paths that starts at the primary cache location.
    The replication algorithm is incremental and follows the same algorithmic steps of the cache distribution algorithm. In the replication process, the objective is to find a replication destination for each VM without creating loops of length smaller than or equal to the resiliency level k. A replication loop is any closed path that revisits the same DF. In Fig. 3, the replication loop $DF1 : VM1 \rightarrow DF2 : VM1 \rightarrow DF3 : VM1 \rightarrow DF1 : VM1$ does not violate the resiliency level 2. A loop $DF1 : VM1 \rightarrow DF2 : VM1 \rightarrow DF1 : VM2$, although spans different VMs, will violate the resiliency level 2.
Similarly to solving the DCDP, the replication algorithm is as follows:

– The warm-starting heuristic matches the VMs of each DF to the VMs of the closest DF. A R-R heuristic is applied to spread the matching across the destination VMs. The loop avoidance is the only criteria in the construction, no balancing of replication load is considered. The replication map in Fig. 3 might result from application of the R-R heuristic. Replication guidelines to make better network resource usage can be added as well (e.g., local replication).
– The CP-based re-sizable decomposition iterates, as in the cache distribution problem, by selecting a sub-problem with the most loaded VM and least loaded VM ($decSize = 1$) that can share replication load. Next, it builds a

CP model of the problem, then changes the current replication map based on the obtained CP solution. A replication deviation test is used to interrupt the replication load distribution process as in the cache distribution algorithm. From Fig. 3, a straight forward input problem can be composed of $DF2 : VM1$ and $DF4 : VM1$ (suppose these 2 VMs can share replication load). Then the CP sub-model is built with the most loaded and least load $DF2 : VM1$ and $DF4 : VM1$. A solution to the CP sub-model that minimizes the replication load deviation could be to move the replica of $DF1 : VM1$ or $DF1 : VM2$ to $DF4 : VM1$.

The replication map is required prior to the assignment of the replicas. The whole cache distribution algorithm (with the replica) is to run sequentially/parallel the primary cache distribution and the replication map building algorithm, then use the replication map to find the locations of the different replicas.

3.7   Discussions

In the development process of the solution above we tried two other alternatives: (1) use standalone Integer Linear Programming (ILP) and CP models, (2) replace the CP sub-model with an ILP sub-model in the decomposition. In the former, we could not meet the run-time requirements of the problem (sub seconds per iteration). The size of the model is too large to find any feasible solution within the specified run-time budget. In the latter, we obtained comparable solutions for some small decomposition size values, but the run-time was higher compared to CP. The weakness of the ILP model was due to the $|VMs| \times |VMs|$ binary constraints to measure and limit the load distance between all the pairs of $|VMs|$.

4   Computational Results

In this section, we present some performance assessments of the proposed solution approach. For our experiments, we used randomly generated instances with realistic properties (size and structure) as proxies for the target instances. Each instance is characterized by a set of DFs, a set of VMs, and a set of data caches. The VMs are randomly distributed across the different DFs, and the sizes of the data caches are randomly generated using two different statistical distributions. In the first distribution the sizes of the caches are uniformly distributed in one contiguous interval. In the second distribution two disjoint sub-intervals, distant from each other, are used to generate caches with larger size discrepancy. The first column of Tables 1 and 2 refer to the properties of these two distributions, where $\bar{s}$ is the mean size value and σ is the standard deviation of the cache size distribution. In total, we generated 160 instances divided into 2 IMDG datasets: small ($50 DFs$, 1,000 VMd, 50,000 caches) and large (100 DFs, 2,000 VMs, 100,000 caches). In each table, there are 4 different intervals (megabyte) for cache distributions referred to as $CacheSize$. For each interval, 10 instances are generated.

Table 1. Small dataset: 50 DFs, 1,000 VMs, and 50,000 caches. The sizes of data caches are uniformly drawn in the intervals of first column.

CacheSize	Small dataset					Large dataset				
$\bar{s}, \sigma$	R-R		Decomposition			R-R		Decomposition		
	RT	Dev	#It	RT	Dev	RT	Dev	#It	RT	Dev
[10...30] 20.5, 5.7	0.63	0.81	266	1.94	0.31	2.85	0.87	668	6.49	0.30
[10...50] 30.5, 11.5	0.64	0.73	205	1.70	0.32	3.20	0.86	533	6.90	0.34
[10...80] 45.5, 20.2	0.61	0.92	273	2.09	0.30	2.44	0.72	486	5.48	0.21
[10...120] 65.4, 31.7	0.62	1.05	302	2.34	0.29	2.85	0.92	534	6.52	0.29

The size of the CP-model in the decomposition can be dynamically changed to fit any solution objective. In fact, it can be changed from one iteration to another when solving the same instance. The higher the size of the decomposition is, the harder the model becomes to solve in practice, and the fewer the required number of iterations to solve the problem is. In order to meet the Coherence sub-seconds per iteration runtime and synchronization requirements, we set the decomposition size to 1. The replication map algorithm is run only when the cloud infrastructure experiences physical changes, i.e., VMs are added/removed and/or failures happened. In our experiments, we set the number of replicas to 1. In practice, the number of replicas has low impact on the performance of the whole solution algorithm.

All experiments were run with an Oracle proprietary CP solver on a 2.5 GHz Intel Processor with 8 GByte main memory. In Tables 1 and 2 we measure the run-time (RT in seconds) and the average deviation (Dev in megabytes) values of the solutions provided by the R-R heuristic and the RT, Dev, and number of iteration (#It) of the provided solutions by the CP-based decomposition (Recall that the CP-based decomposition is warm-started with the R-R heuristic).

In Table 1, we see clearly that the R-R heuristic alone finds high quality solutions within short runtime. The increase in the size of the instances did not alter the quality of the solutions. Because the standard deviations (σ) of the cache distributions are not high, the R-R heuristic easily compensates the load shifts by the forward and backward distribution mechanism. The CP-based decomposition reduced the load deviations of all the instances in Table 1 by a factor of up to ~73 %. After an average number of iterations equals to ~25 % of |VMs| and a few seconds run-time, the CP-based decomposition considerably improves the solutions found by R-R. The number of iterations indicates that the CP-based decomposition, on-average, involved at most 50 % of the VMs (#It equal to 25 % of |VMs|, and each iteration involves a pair of VMs). The other

Table 2. Large dataset: 100 DFs, 2,000 VMs, and 100,000 caches. The sizes of 98 % of data caches are uniformly drawn in the first part of the intervals of first column, 2 % in the second part of the intervals.

CacheSize $\bar{s}, \sigma$	Small dataset					Large dataset				
	R-R		Decomposition			R-R		Decomposition		
	RT	Dev	#It	RT	Dev	RT	Dev	#It	RT	Dev
[10...30][60...90] 21.6, 11.75	0.63	36.28	1008	4.40	0.35	2.85	36.10	2076	11.83	0.32
[10...50][100...150] 32.4, 20.7	0.64	60.49	1164	5.85	0.34	3.20	60.51	2317	13.70	0.33
[10...80][160...240] 48.6, 34.5	0.62	96.58	1348	7.75	0.27	2.44	96.50	2681	18.86	0.34
[10...120][240...360] 70.19, 52.8	0.61	144.85	1376	8.88	0.30	2.85	144.56	2764	20.62	0.33

50 % of VMs are assigned caches by the R-R heuristic and not rebalanced by the CP-based decomposition.

In Table 2 the R-R alone cannot find good quality solutions anymore. The CP-based decomposition drastically reduced the load deviations of all the instances by a factor of up to ~99.8 %. Obviously the discrepancies in the cache distributions created more challenging instances to solve. Both the number of iterations and the run-time required by the CP-based decomposition are slightly higher compared to the previous runs with the one interval cache distributions. The number of iterations in this case increased to more than 100 % of |VMs|, which means that the CP-based decomposition, on-average, involved each of the VMs at least twice.

5   Incrementality Advantages

Our proposed decomposition is an incremental approach that optimizes the load deviation by performing local load re-distributions across the VMs. Suggestions of assignments are formulated by the CDS at each iteration of the CP-based decomposition. The CDM can either implement the suggestions, or decide to ignore them and interrupt CDS for some reason, e.g., congested network infrastructure. After an interruption, if no major change occurs in the IMDG state information, then the CP-based decomposition is likely to regenerate the sub-model of the interrupted problem. As opposed to a single call to CDS, this type of incrementality gives high advantage to our solution in complex and highly dynamic IDMG systems where caches grow in an unpredictably way, and failures occur more frequently.

6   Conclusions

We presented a CP-based solution approach for solving the Oracle Coherence DCDP. The proposed solution is composed of a re-sizable CP-based decomposition method warm started with a R-R heuristic. The CP-based decomposition consists of iteratively creating and solving CP sub-models to globally improve the cache load balance. The same cache distribution solution approach is generalized to solve the replication problem for achieving k resiliency in Coherence.

Extensive computational experiments are performed to assess the quality of the obtained solutions. The results clearly show the added value of the proposed CP-based solution for providing optimized incremental load distribution in Oracle Coherence. Solutions for large-scale Coherence instances are obtained within a few seconds of runtime.

There remains interesting future directions to explore including the trade-off between the decomposition size and the runtime. Another promising research direction is the parallelization of the solution. The proposed decomposition approach is naturally parallel as multiple CP-models can be created and executed in parallel with little overhead.

References

1. Ardagna, D., Casale, G., Ciavotta, M., Pérez, J.F., Wang, W.: Quality-of-service in cloud computing: modeling techniques and their applications. J. Internet Serv. Appl. **5**(1), 1–17 (2014)
2. Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E.K., Moatti, Y., Lorenz, D.H.: Guaranteeing high availability goals for virtual machine placement. In: IEEE Distributed Computing Systems (ICDCS), pp. 700–709 (2011)
3. Chen, K., Hu, C., Zhang, X., Zheng, K., Chen, Y., Vasilakos, A.V.: Survey on routing in data centers: insights and future directions. IEEE Netw. **25**(4), 6–10 (2011)
4. Hermenier, F., Demassey, S., Lorca, X.: Bin repacking scheduling in virtualized datacenters. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 27–41. Springer, Heidelberg (2011)
5. Hermenier, F., Lorca, X., Menaud, J., Muller, G., Lawall, J.L.: Entropy: a consolidation manager for clusters. In: VEE, pp. 41–50. ACM (2009)
6. Lubinski, T.: Detecting and alerting on fault conditions in an oracle coherence distributed caching system. Oracle documentation (2011)
7. Mehta, D., O'Sullivan, B., Simonis, H.: Comparing solution methods for the machine reassignment problem. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 782–797. Springer, Heidelberg (2012)
8. Pesant, G., Régin, J.-C.: SPREAD: a balancing constraint based on statistics. In: Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 460–474. Springer, Heidelberg (2005)
9. Rai, A., Bhagwan, R., Guha, S.: Generalized resource allocation for the cloud. In: Proceedings of the Third ACM Symposium on Cloud Computing, p. 15. ACM (2012)
10. Régin, J.C., Rezgui, M.: Discussion about constraint programming bin packing models. In: AI for Data Center Management and Cloud Computing (2011)

11. Ruzzi, J.: Oracle coherence getting started guide, release 3.6. Oracle documentation (2010)
12. Schaus, P.: Solving balancing and bin-packing problems with constraint programming. These de doctorat, Université catholique de Louvain (2009)
13. Wolke, A., Tsend-Ayush, B., Pfeiffer, C., Bichler, M.: More than bin packing: dynamic resource allocation strategies in cloud data centers. Inf. Syst. **52**, 83–95 (2015)
14. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. J. Internet Serv. Appl. **1**(1), 7–18 (2010)

Computational Sustainability Track

Online HVAC-Aware Occupancy Scheduling with Adaptive Temperature Control

BoonPing Lim[(✉)], Hassan Hijazi, Sylvie Thiébaux, and Menkes van den Briel

NICTA, The Australian National University, Canberra, Australia
`boonping.lim@anu.edu.au`

Abstract. Heating, ventilation and air-conditioning (HVAC) is the largest consumer of electricity in commercial buildings. Consumption is impacted by group activities (e.g. meetings, lectures) and can be reduced by scheduling these activities at times and locations that minimize HVAC utilization. However, this needs to preserve occupants' thermal comfort and be responsive to dynamic information such as new activity requests and weather updates. This paper presents an *online* HVAC-aware occupancy scheduling approach which models and solves a joint HVAC control and occupancy scheduling problem. Our online algorithm greedily commits to the best schedule for the latest activity requests and notifies the occupants immediately, but revises the entire future HVAC control strategy each time it considers new requests and weather updates. In our experiments, the quality of the solution obtained by this approach is within 1 % of that of the clairvoyant solution. We incorporate *adaptive comfort temperature control* into our model, encouraging energy saving behaviors by allowing the occupants to indicate their thermal comfort flexibility. In our experiments, the integration of adaptive temperature control further generates up to 12 % of energy savings when a reasonable thermal comfort flexibility is provided.

1 Introduction

Heating, ventilation and air-conditioning (HVAC) dominates the energy consumption of commercial buildings, accounting for roughly 40 % of the total building electricity consumption per annum [11,28]. With rising energy costs and increasingly stringent regulatory environments, improving the energy efficiency of HVAC operations in buildings has become an important issue.

Recent studies show that energy-oriented occupancy scheduling can lead to significant savings in energy consumption [5,19–24,26,27]. The idea is to *proactively* control occupancy in commercial offices and university buildings by scheduling energy-hungry activities such as meetings, workshops, lectures and exams, at times and locations that are favorable from an energy standpoint. Lim et al.'s "HVAC-aware" occupancy scheduling approach implements this idea by solving the *joint* HVAC control and occupancy scheduling problem [21,22], which consists in simultaneously optimizing the times and locations of the various activities and the HVAC control parameters at each time and building zone.

By exploiting the synergy between HVAC control and occupancy scheduling, this approach achieves a much higher rate of energy savings than works that are based on (data-driven) black-box models of the HVAC control [5,20] or that minimize energy consumption proxies (e.g. number of rooms used) [24,27].

Unfortunately, with few exceptions [20], previous works focus on off-line scheduling, and assume that all activities to schedule and other parameters such as the weather forecast are known in advance. Existing approaches also adopt a fixed comfort temperature control, keeping the allowable temperature of occupied locations strictly within narrow bounds (e.g. 21°-23°C). Although both settings generate energy-efficient schedules, they nevertheless limit the practicability of the models in the real-world, and prevent further energy savings that could be obtained with more flexible temperature bounds. This paper presents two novel contributions which address these shortcomings. First, we extend Lim et al.'s HVAC-aware occupancy scheduling approach to process activity requests in an *on-line* manner. Second, we encourage energy saving behavior by allowing the occupant to indicate their thermal comfort flexibility, and use it in a principled way to introduce *adaptive comfort temperature control* in our model.

In more detail, we propose an on-line approach that models and solves the joint HVAC control and occupancy scheduling problem. Our on-line algorithm greedily optimizes (and commits to) the times and locations for the latest requests, leaving the rest of the future schedule fixed but revising the entire future HVAC control strategy. This ensures that whilst participants are instantly notified of the scheduled time and location for their requested activity, the HVAC control is constantly re-optimized and adjusted to the full schedule and weather updates. Our experiments demonstrate that the quality of the online solution is, on average, within 1 % of that of the solution returned by the original clairvoyant HVAC-aware algorithm [22].

Adaptive comfort temperature control shifts away from fixed indoor comfort bands towards wider temperature operating bands. Recent work [1] shows that even a narrow variation of comfort temperatures can achieve significant energy savings. We introduce the notion of thermal comfort flexibility in our model by allowing occupants to indicate their level of tolerance to temperature fluctuation in the form of (a) a threshold limiting the probability of temperature violation, and (b) the maximum deviation allowed at any time. We then solve a robust optimization model which provide these probabilistic guarantees. Our experiments show that, when occupants are reasonably flexible, the integration of adaptive temperature control generates up an extra 12 % of energy savings.

As an additional advantage, adaptive temperature control reduces the constrainedness of our online scheduling and control problem. This can make the problem solvable whereas fixed temperature bounds cannot be met, which often occurs for instance when a late request needs to be scheduled in the immediate future in a room whose current temperature is far away from the comfort band. In our experiments, adaptive temperature control solves 73 % of the 700 instances that are unsolvable under the fixed temperature control model.

To summarize, the main contributions of the paper are: (a) an efficient online model for the joint HVAC control and occupancy scheduling problem, (b) a new notion of thermal comfort flexibility in energy-aware scheduling, (c) experiments showing substantial energy reduction and improvement of solution feasibility over the state-of-the-art.

2 Related Work

Our work differs from previous work given its focus on: (i) a joint HVAC control and occupancy scheduling model which handles dynamically arriving scheduling requests, and (ii) an adaptive temperature control approach that allows the occupant to specify their thermal comfort flexibility. Existing works on energy-aware occupancy scheduling [5, 21–24, 26, 27] focus on offline scheduling, and assume fixed comfort temperature setpoints. In reality, scheduling requests can arrive at any time of the day using existing room booking systems. A recent survey shows that 56 % of meeting requests were made within 1 day before the actual meeting day [20]. Thus, the ability to handle impromptu requests is crucial. Moreover, the ability to update HVAC control following a change in forecast is also essential.

Kwak et al. [19, 20] propose an online stochastic MILP model to schedule meetings. Their work calculates energy consumption based on historical data and exploits flexibility in the time and location at which a meeting can take place. However, it does not optimize HVAC control, nor does it take thermal comfort flexibility into account. Our results show that combining meeting scheduling with HVAC control, and enabling adaptive temperature control based on occupant thermal comfort flexibility, significantly impacts energy savings.

Conventionally, room temperature is maintained within strict comfort bounds while occupied. Such control is not the most effective, since the HVAC system tries to achieve fixed temperature setpoints regardless of ambient conditions or the comfort levels of the individual occupants. More recent work enables adaptive thermal comfort control [1, 6, 7, 18, 25, 31, 32], exploiting the observation that when occupants have some form of input to the control, their subjective view of comfort changes and they are more willing to accept wider operating conditions than those mandated by traditional comfort models. For example, when controlling the emissivity of dynamic windows to reduce HVAC consumption in a smart home, Ono et al. [25] allow for temperature bound violations, but limit their probability using chance constraints with occupant-specified thresholds. Inspired by these works, we introduce the notion of thermal comfort flexibility into our scheduling model. We incorporate occupants' tolerance level as an input, allowing the scheduler to identify the best location and time slots that optimize energy saving while satisfying occupant thermal comfort.

Energy-oriented scheduling has gained more attention in recent years due to the significant cost saving opportunities. Ifrim et al. [17] present a MIP-based energy-price savings scheduling model to reduce cost in production scheduling. Dupont et al. [8] use CP to develop an energy aware framework for virtual machine placement in cloud-based data centers. Scott et al. [29] describe an

online stochastic MILP to schedule home appliances based on real-time pricing. Most works focus on energy-aware scheduling in production lines, data centers and residential buildings whilst our work specifically targets energy-efficient scheduling in the smart building space, which is dominated by HVAC consumption.

3   Online Occupancy Scheduling

This section presents our online occupancy scheduling problem. We start by describing the scheduling setting and our notations. We then cover the scheduling constraints and variables which, later on in Sect. 4, will interact with the HVAC control model to form a more complex joint scheduling and control model. We formulate our model as a mixed-integer program (MIP). It can be solved using a MIP solver, or when scaling up to problems of practical size, by combining MIP with large neighborhood search (LNS) as explained in [22].

In our online setting, the scheduler runs recurrently and each run is called an *online session*. Each online session $i \in I$ starts at time τ_i and ends before the next session starts at time τ_{i+1}. The scheduling and control model discretizes time into a set K of *time steps*. Each time step $k \in K$ starts at time t_k. Two consecutive time steps k and $k+1$ are separated by a fixed duration $t_{k+1} - t_k = \boldsymbol{\Delta_t} \in \mathbb{R}^+$. Each on-line session i considers a horizon of $\boldsymbol{n}$ time steps $K(i) = \{k(i), \ldots, k(i) + \boldsymbol{n} - 1\}$ where $k(i)$, the first time step in that horizon, is the least time step in K such that $t_{k(i)} \geq \tau_i$.

Let L be the set of locations (or, interchangeably, zones) in the building, and P be a set of participants. An activity request m is a tuple $\langle \boldsymbol{a}_m, K_m, L_m, P_m, \boldsymbol{d}_m, F_m \rangle$ where $\boldsymbol{a}_m \in \mathbb{R}^+$ is the request arrival time, $K_m \subseteq K$ is the set of time steps at which the activity is permitted to <u>start</u> in the future (for each $k \in K_m, \boldsymbol{a}_m < t_k$), $L_m \subseteq L$ is the set of locations at which the activity is permitted to take place, $P_m \subseteq P$ is the set of attendees for the activity, $\boldsymbol{d}_m \in \mathbb{N}$ is the activity duration (number of time steps), and F_m represents the comfort temperature flexibility parameters which will be explained in Sect. 4.4. Note that the sets K_m and L_m can be used to encode a variety of situations, such as room capacity requirements, availability of special equipment such as video conferencing, time deadlines for the activity, and attendee availability constraints. We write $\mathcal{C}(M)$ for the set of attendee conflicts w.r.t. a set of requests M; each conflict C is a subset of requests, each pair of which has at least one attendee in common: $\mathcal{C}(M) = \{C \subseteq M \mid \forall m, m' \in C, P_m \cap P_{m'} \neq \emptyset\}$.

To account for all activities that have been scheduled so far, we maintain a master schedule S as a set of triples $\langle m, l, k \rangle$ storing the activity request id m, the assigned location l, and the time step k at which m is scheduled to start. At each online session i, the scheduler schedules the new activity requests $N(i)$ which have been received since the start of session $i - 1$, i.e., each $m \in N(i)$ satisfies $\tau_{i-1} < \boldsymbol{a}_m \leq \tau_i$. It also needs to consider, without modifying them, the set $Q(i)$ of ongoing activities and future activities that were scheduled during previous sessions: $Q(i) = \{m \mid \exists \langle m, l, k \rangle \in S \text{ such that } k + \boldsymbol{d}_m - 1 \geq k(i)\}$.
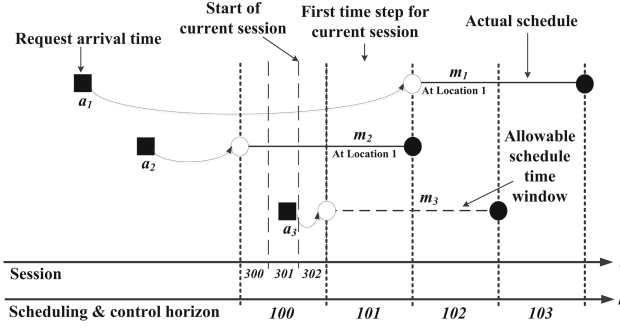
Fig. 1. Online scenario

So overall, the set of activities to consider at session i is $M(i) = N(i) \cup Q(i)$. To simplify the scheduling model below, we assume that for each pre-scheduled request $m \in Q(i)$ such that $\langle m, l, k \rangle \in S$, the set of permissible locations is reduced to $L_m = \{l\}$, and the set of permissible start time steps is reduced to the scheduled start time k or the first time step $k(i)$ of the session, which ever occurs last, i.e. $K_m = \{\max(k, k(i))\}$. For consistency, the meeting duration $\boldsymbol{d}_m$ is decremented by $k(i) - k$; this is only needed later in Sect. 4.4 for Eq. (23).

Figure 1 shows a scenario example featuring three requests m_1, m_2 and m_3 with arrival times $\boldsymbol{a}_1, \boldsymbol{a}_2$ and $\boldsymbol{a}_3$, respectively. The set of locations is $L = \{\boldsymbol{l}_1, \boldsymbol{l}_2\}$. The dash vertical lines show the start of the sessions, and the dotted vertical lines delimit the time steps. In this instance, the scheduler runs every 10 min and the time steps are 30 min long. At the start of session $i = 302$, requests m_1 and m_2 have already been scheduled and m_3 is a new request, hence $N(302) = \{m_3\}$, $Q(302) = \{m_1, m_2\}$. The master schedule is $S = \{\langle m_1, \boldsymbol{l}_1, 102 \rangle, \langle m_2, \boldsymbol{l}_1, 100 \rangle\}$, and the first time step of the new session is $k(302) = 101$. The set of permissible locations and start time steps for the new request are $K_3 = \{101, 102\}$ and $L_3 = \{\boldsymbol{l}_1, \boldsymbol{l}_2\}$ ($\boldsymbol{l}_1$ will be ruled out by the scheduler). Those of the pre-existing requests are reduced as follows: $L_1 = \{\boldsymbol{l}_1\}, K_1 = \{102\}, L_2 = \{\boldsymbol{l}_1\}$ and $K_2 = \{101\}$.

We are now ready to describe our scheduling constraints and variables for online session i. The main scheduling variable is the boolean decision variable $x_{m,l,k}$ which is true iff request $m \in M(i)$ is scheduled to take place at zone $l \in L_m$ starting at time slot $k \in K_m$. We also introduce the variables $y_{m,l,k}$ which is true iff activity m is scheduled to occupy location l at time step k, $z_{l,k}$ which is true iff zone l is occupied at time step k, and $pp_{l,k}$ which indicates the number of people in zone l at time step k. These variables will be used by the HVAC control part of the model in 4.

The scheduling constraints are the following. Constraints (1) ensure that all requests are scheduled exactly once within the allowable start times and locations. Constraints (2) define the $y_{m,l,k}$ variables. Constraints (3) state that no more than one activity can occupy a location at any time and define the $z_{l,k}$ variables. Observe that the right hand side of these constraints is either zero or one, which limits the number of activities to at most one. Also, when the left

hand side equals one then the zone must be occupied. Constraints (4) determine the number $pp_{l,k}$ of occupants at each location and time step, and finally constraints (5) ensure that activities with at least one attendee in common cannot be scheduled in parallel. Once a new request $m \in N(i)$ has been scheduled, the master schedule S is updated by adding the 3-tuple $\langle m, l, k \rangle$ for which $x_{m,l,k} = 1$.

$$\sum_{l \in L_m, k \in K_m} x_{m,l,k} = 1 \quad \forall m \in M(i) \tag{1}$$

$$\sum_{\substack{k' \in K_m: \\ l \in L_m, \ k - \boldsymbol{d}_m + 1 \leq k' \leq k}} x_{m,l,k'} = y_{m,l,k} \quad \forall m \in M(i), l \in L, k \in K(i) \tag{2}$$

$$\sum_{m \in M(i)} y_{m,l,k} \leq z_{l,k} \quad \forall l \in L, k \in K(i) \tag{3}$$

$$\sum_{m \in M(i)} y_{m,l,k} \times |P_m| = pp_{l,k} \quad \forall l \in L, k \in K(i) \tag{4}$$

$$\sum_{m \in \nu, l \in L_m} y_{m,l,k} \leq 1 \quad \forall k \in K(i), \nu \in \mathcal{C}(M(i)) \tag{5}$$

4   HVAC Control Model

This section covers the HVAC control model and the adaptive temperature control approach. We describe the HVAC control aspects starting with the objective function we consider, the effect of the control on the building thermal dynamics, and the fixed temperature bounds – we refer the reader to [21] for a more detailed treatment. We subsequently extend the model with adaptive temperature control to further maximize energy savings.

4.1   Variable-Air-Volume Systems

Following Goyal et al. [14,15], we focus on commercial buildings with variable-air-volume (VAV) based HVAC systems, which serve over 30 % of the commercial building floor space in the United States [9]. A schematic of a VAV-based HVAC system with two VAV boxes connected to two building zones is shown in Fig. 2.

The air handling unit (AHU) supplies conditioned air to the VAV boxes. The AHU consumes energy when mixing outdoor air with return air and cooling it to the pre-set conditioned air temperature T^{CA} [12.8 °C]; it consumes less energy when the outdoor air temperature T^{OA} is closer to T^{CA}. Each VAV box consumes energy when regulating the supply air temperature T^{SA} and the supply air flow rate a^{SA} to keep the zone temperature T within comfort bounds; in particular, it may need to reheat the conditioned air. Finally, the supply fan at the AHU consumes energy to maintain a constant air pressure through the supply duct; it may speed up or slow down depending on air flow rates used by the VAV boxes.
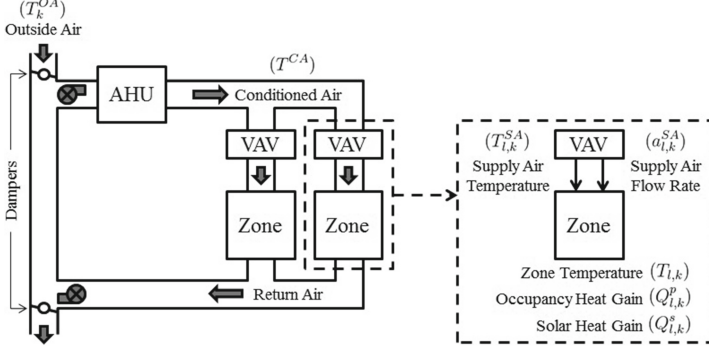
Fig. 2. VAV-based HVAC system.

We focus on control strategies that can be applied to each VAV box. For such strategies, the key HVAC decision variables are the supply air flow rate $a_{l,k}^{SA}$ and temperature $T_{l,k}^{SA}$ at each zone/location $l \in L$ and time step $k \in K$. We determine an optimal control for these variables, given the occupancy schedule and the bounds on supply air temperature, supply air flow rate, and room temperature during vacant and occupied periods.

4.2   Objective Function

Specifically, our goal is to generate energy-efficient schedules that minimize the energy use of air-conditioning, re-heating and fan operations of the HVAC. Thus, the objective function for online session i is the following.

$$\text{minimize} \sum_{k \in K(i)} \left(p_k^{cond} + p_k^{fan} + \sum_{l \in L} p_{l,k}^{heat} \right) \times \boldsymbol{\Delta_t} \qquad (6)$$

where

$$p_k^{cond} = \boldsymbol{C}^{pa} \left(T_k^{OA}(i) - \boldsymbol{T}^{CA} \right) \sum_{l \in L} a_{l,k}^{SA} \quad \forall k \in K(i) \qquad (7)$$

$$p_k^{fan} = \boldsymbol{\beta} \sum_{l \in L} a_{l,k}^{SA} \quad \forall k \in K(i) \qquad (8)$$

$$p_{l,k}^{heat} = \boldsymbol{C}^{pa} (T_{l,k}^{SA} - \boldsymbol{T}^{CA}) a_{l,k}^{SA} \quad \forall l \in L, k \in K(i) \qquad (9)$$

Constraints (7)-(9) determine the values of the variables $p_k^{cond}, p_k^{fan}, p_{l,k}^{heat}$, which respectively represent the energy consumed by the AHU for conditioning, by the supply fan for maintaining air pressure, and by the VAV box for reheating the conditioned air. In constraint (7), we assume that online session i uses the latest update $T_k^{OA}(i)$ available for the outdoor temperature forecast at each time step

k. The coefficients in these constraints are the fan power coefficient $\boldsymbol{\beta}$ (0.65), and the heat capacity of air at constant pressure $\boldsymbol{C}^{pa}$ (1.005 kJ/kg·K).

$$\boldsymbol{T}^{CA} \leq T_{l,k}^{SA} \leq \overline{\boldsymbol{T}}^{SA} \quad \forall l \in L, k \in K(i) \tag{10}$$

$$\underline{\boldsymbol{a}}^{SA} \leq a_{l,k}^{SA} \leq \bar{\boldsymbol{a}}^{SA} \quad \forall l \in L, k \in K(i) \tag{11}$$

Moreover, constraints (10) and (11) ensure that the supply air temperature and the air flow rate are bounded by the HVAC operational capacity. The supply air temperature $T_{l,k}^{SA}$ may range from that of the conditioned air $\boldsymbol{T}^{CA}$ (12.8 °C), up to $\overline{\boldsymbol{T}}^{SA}$ (40 °C) if the air is reheated at the VAV box. The air flow rate $a_{l,k}^{SA}$ can fluctuate between $\underline{\boldsymbol{a}}^{SA}$ (0.108 kg/s) and $\bar{\boldsymbol{a}}^{SA}$ (5.0 kg/s), where the lower bound is determined by the ASHRAE ventilation standard and the upper bound is reached when the VAV dampers are fully open.

4.3 Building Thermal Dynamics

Next, we want our control to appropriately constrain zone temperatures. The first step to do this is to introduce a new variable $T_{l,k}$ representing the temperature at each zone and time step, and model the effects of the HVAC control on this zone temperature. To capture the building thermal dynamics, we adopt a computationally efficient lumped RC-network [12–14] which incorporates the thermal resistance and capacitance of each zone and between adjacent zones, the latest available forecast of the solar gain $Q_{l,k}^s(i)$, and the internal heat gain $Q_{l,k}^p$ generated by the occupants at each zone. The latter is directly proportional to the number of occupants $pp_{l,k}$ scheduled to be at the zone by the online scheduler – this is one of the variables via which the scheduling and control models interact. We use a discrete-time linear model

$$T_{l,k+1} = f_l(T_{l,k}, u_{l,k}, v_{l,k}) \quad \forall l \in L, k \in K(i) \tag{12}$$

where $u_{l,k} = [a_{l,k}^{SA}, T_{l,k}^{SA}, pp_{l,k}]$ is the vector of controllable variables, and $v_{l,k} = [Q_{l,k}^s(i), T_k^{OA}(i)]$ is the vector of exogenous inputs. With this model, the HVAC control is optimized over the entire horizon $K(i)$. E.g., the optimal control could activate the HVAC at night to benefit from the low outside night temperature to pre-cool a room for an early morning meeting. See [21] for details.[1]

4.4 Adaptive Temperature Control

Having modeled the effect of the HVAC control on the zone temperatures $T_{l,k}$, we are now ready to ensure that the HVAC fulfills its main role of keeping these zone temperatures within appropriate comfort bounds. In the fixed comfort

---

[1] Both Lim et al. [21,22] and our experiments use a more complex state vector which not only includes the zone temperatures $T_{l,k}$ but also the temperature of the interior walls. For readability reasons, we abstract from these extra state variables in our exposition above.

bound model found in much of the literature, when a zone is occupied, the zone temperature must lie within a specified comfort interval $[\underline{\boldsymbol{T}}, \overline{\boldsymbol{T}}]$ ([21 °C, 23 °C]). When the zone is empty, its temperature can fluctuate more freely within $[\underline{\boldsymbol{T}}^{\emptyset}, \overline{\boldsymbol{T}}^{\emptyset}]$ (16 °C, 28 °C]). These bounds can be set to reflect individual building guidelines. As shown in [21], maintaining temperature within these fixed bounds can be achieved by adding constraints (13) to our model. In these constraints, the HVAC model interacts with the scheduling model via the variables $z_{l,k}$ that indicate whether or not location l is occupied at time step k. The constants $\underline{\boldsymbol{T}}^g$ and $\overline{\boldsymbol{T}}^g$ denote the gap between the occupied and unoccupied temperature lower and upper bounds.

$$\underline{\boldsymbol{T}}^{\emptyset} + \underline{\boldsymbol{T}}^g z_{l,k} \leq T_{l,k} \leq \overline{\boldsymbol{T}}^{\emptyset} - \overline{\boldsymbol{T}}^g z_{l,k} \quad \forall l \in L, k \in K(i) \tag{13}$$

In the present paper, we generate additional energy savings by departing from these fixed comfort bounds. We adopt a flexible temperature bound model, in which the comfort interval is dynamically configured through input parameters reflecting the flexibility of occupants. Specifically, the input parameters we consider for an activity request m are $F_m = \langle \boldsymbol{T}_m^u, \boldsymbol{\alpha}_m, \boldsymbol{p}_m \rangle$ and are such that the HVAC control will guarantee: (a) that the zone temperature will never exceed $[\underline{\boldsymbol{T}} - \boldsymbol{T}_m^u, \overline{\boldsymbol{T}} + \boldsymbol{T}_m^u]$ at any point during the activity and (b) that with probability at least $\boldsymbol{p}_m$, the cumulative temperature violation during the activity will be bounded by $\boldsymbol{\alpha}_m$. The parameter $\boldsymbol{\alpha}_m$ is equivalent to the duration for which the occupant would be willing to let the temperature deviation be $\boldsymbol{T}_m^u$. Figure 3 illustrates these concepts. In this example, activity m occupies location l for 3 times steps. The occupant is prepared to accept a maximal deviation (of up to 3°) from the default comfort bounds (i.e. [18 °C, 26 °C]), but also wants the cumulative violation to remain within acceptable bounds (the equivalent of 20 min at 3°) with high probability (0.9). This is achieved by setting $\boldsymbol{T}_m^u = 3$, $\boldsymbol{\alpha}_m = 20$, and $\boldsymbol{p}_m = 0.9$.

Let m be a meeting scheduled to start at time step $j \in K_m$ in location l. To formalize these concepts, we introduce the following slack variables in the model $\underline{\boldsymbol{T}}_{m,k}^s \in [0, \boldsymbol{T}_m^u]$ and $\overline{\boldsymbol{T}}_{m,k}^s \in [0, \boldsymbol{T}_m^u]$, for $k \in K(i)$. These variables represent our unknown temperature violations above and below the default bounds $[\underline{\boldsymbol{T}}, \overline{\boldsymbol{T}}]$.
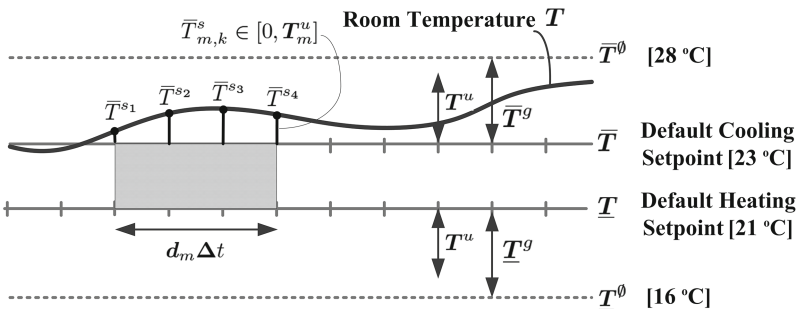

Fig. 3. Adaptive Temperature Control

Based on these variables, the first guarantee we want to provide can be written as the adaptive counterpart of the fixed temperature bound constraints (13).

$$\underline{\boldsymbol{T}}^{\emptyset} + \underline{\boldsymbol{T}}^g z_{l,k} - \underline{\boldsymbol{T}}^s_{m,k} \leq T_{l,k} \leq \overline{\boldsymbol{T}}^{\emptyset} - \overline{\boldsymbol{T}}^g z_{l,k} + \overline{\boldsymbol{T}}^s_{m,k} \tag{14}$$

The second guarantee is about bounding the cumulative temperature violation, this can be formulated as follows,

$$\sum_{k=j}^{j+\boldsymbol{d}_m-1} \left(\underline{\boldsymbol{T}}^s_{m,k} + \overline{\boldsymbol{T}}^s_{m,k} \right) \boldsymbol{\Delta}_t \leq \boldsymbol{\alpha}_m \boldsymbol{T}^u_m \tag{15}$$

To implement a probabilistic version of this constraint, we introduce independent uniformly distributed random variables $\rho_{m,k} \in [-1,1]$, which represent the noise in our temperature violation, transforming constraints (15) into,

$$\sum_{k=j}^{j+\boldsymbol{d}_m-1} \left(\underline{\boldsymbol{T}}^s_{m,k} + \overline{\boldsymbol{T}}^s_{m,k} - \rho_{m,k} \right) \boldsymbol{\Delta}_t \leq \boldsymbol{\alpha}_m \boldsymbol{T}^u_m \tag{16}$$

We then resort to results from the Robust Optimization literature [2–4, 10, 16] to be able to offer the following probabilistic guarantee,

$$Pr \left(\sum_{k=j}^{j+\boldsymbol{d}_m-1} \left(\underline{\boldsymbol{T}}^s_{m,k} + \overline{\boldsymbol{T}}^s_{m,k} - \rho_{m,k} \right) \boldsymbol{\Delta}_t \leq \boldsymbol{\alpha}_m \boldsymbol{T}^u_m \right) \geq \boldsymbol{p}_m \tag{17}$$

where $Pr\left(f_{\boldsymbol{\rho}}(x) \leq 0\right)$ denotes the probability of satisfying constraint $f_{\boldsymbol{\rho}}(x) \leq 0$ given the uncertainty created by the random variables $\boldsymbol{\rho}$. In particular, based on [2, Theorem 3.], we can offer the above probabilistic guarantee by enforcing the following constraint

$$\sum_{k=j}^{j+\boldsymbol{d}_m-1} \rho^2_{m,k} \leq \boldsymbol{\delta}^2_m, \tag{18}$$

where the the ellipsoid radius $\boldsymbol{\delta}_m$ is linked to the constraint satisfaction probability $\boldsymbol{p}_m$ as follows:

$$\boldsymbol{p}_m \geq 1 - exp(-\boldsymbol{\delta}^2_m/1.5).$$

For instance, a radius of $\boldsymbol{\delta}_m = 2.63$ leads to a constraint satisfaction probability $\boldsymbol{p}_m \geq 0.99$. Furthermore, based on [16, Corollary 1], we can write the following deterministic equivalent of (17) without having to explicitly enforce (18),

$$\sum_{k=j}^{j+\boldsymbol{d}_m-1} \left(\underline{\boldsymbol{T}}^s_{m,k} + \overline{\boldsymbol{T}}^s_{m,k} \right) - |\mathcal{S}| - \sqrt{(\boldsymbol{\delta}^2_m - |\mathcal{S}|) |\overline{\mathcal{S}}|} \leq \boldsymbol{\alpha}_m \boldsymbol{T}^u_m / \boldsymbol{\Delta}_t, \tag{19}$$

where the set $\mathcal{S}$ is described in [16, Proposition 1]. For computational efficiency reasons, this is the approach we adopt in our current implementation.

Since activity locations and start times are not known in advance, we introduce variables $\underline{T}_{l,k}^{\xi}$ (resp. $\overline{T}_{l,k}^{\xi}$) such that $\underline{T}_{l,k}^{\xi} = \underline{T}_{m,k}^{s}$ and $\overline{T}_{l,k}^{\xi} = \overline{T}_{m,k}^{s}$ when activity $m \in M(i)$ occupies location $l \in L$ at time slot $k \in K(i)$, i.e., when $y_{m,l,k} = 1$. In order to accommodate activities that span multiple scheduling horizons, we also introduce the inputs $\boldsymbol{T}_{m}^{prev} = \sum\limits_{k \in K : k < k(i)} (\underline{\boldsymbol{T}}_{m,k}^{s} + \overline{\boldsymbol{T}}_{m,k}^{s})$, which accounts for the amount of cumulative violation consumed before the start of the current session. Recall also from Sect. 3 that meetings that have been scheduled in previous sessions have their start time set $\boldsymbol{K}_{m}$, location set $\boldsymbol{L}_{m}$ and duration $\boldsymbol{d}_{m}$ reduced accordingly when the current session starts. With these notations, the overall adaptive temperature control constraints replacing the fixed temperature constraints (13) in the HVAC control model are the following.

$$\underline{\boldsymbol{T}}^{\emptyset} + \underline{\boldsymbol{T}}^{g} z_{l,k} - \underline{T}_{l,k}^{\xi} \le T_{l,k} \le \overline{\boldsymbol{T}}^{\emptyset} - \overline{\boldsymbol{T}}^{g} z_{l,k} + \overline{T}_{l,k}^{\xi} \quad \forall l \in L, k \in K(i) \tag{20}$$

$$\underline{T}_{m,k}^{s} - \hat{\boldsymbol{T}} \left(1 - y_{m,l,k}\right) \le \underline{T}_{l,k}^{\xi} \le \underline{T}_{m,k}^{s} + \hat{\boldsymbol{T}} \left(1 - y_{m,l,k}\right) \; \forall m \in M(i), l \in L, k \in K(i) \tag{21}$$

$$\overline{T}_{m,k}^{s} - \hat{\boldsymbol{T}} \left(1 - y_{m,l,k}\right) \le \overline{T}_{l,k}^{\xi} \le \overline{T}_{m,k}^{s} + \hat{\boldsymbol{T}} \left(1 - y_{m,l,k}\right) \; \forall m \in M(i), l \in L, k \in K(i) \tag{22}$$

$$\sum_{k=j}^{j+\boldsymbol{d}_{m}-1} \left(\underline{T}_{m,k}^{s} + \overline{T}_{m,k}^{s}\right) - |\mathcal{S}| - \sqrt{\left(\boldsymbol{\delta}_{m}^{2} - |\mathcal{S}|\right) |\overline{\mathcal{S}}|} \le \boldsymbol{\alpha}_{m} \boldsymbol{T}_{m}^{u} / \boldsymbol{\Delta}_{t} - \boldsymbol{T}_{m}^{prev}, \forall m \in M(i), j \in \boldsymbol{K}_{m} \tag{23}$$

$$\underline{T}_{l,k}^{\xi} \le \sum_{m \in M(i)} \boldsymbol{T}_{m}^{u} y_{m,l,k} \quad l \in L, k \in K(i) \tag{24}$$

$$\overline{T}_{l,k}^{\xi} \le \sum_{m \in M(i)} \boldsymbol{T}_{m}^{u} y_{m,l,k} \quad l \in L, k \in K(i) \tag{25}$$

Constraints (20) are the adaptive bound constraints. Constraints (21-22) are the on-off constraints defining the variables $\underline{T}_{l,k}^{\xi}$ and $\overline{T}_{l,k}^{\xi}$ with $\hat{\boldsymbol{T}} = \max\limits_{m \in M(i)} \{\boldsymbol{T}_{m}^{u}\}$. Constraint (23) is the probabilistic constraint on the cumulative temperature violation, taking into account $\boldsymbol{T}_{m}^{prev}$. The last two constraints force the corresponding slack to zero when a location is unoccupied.

5   Experimental Results

5.1   Problem Sets

We analyze our contributions using 9 problem sets with increasing numbers of activities (meetings) and locations (meeting rooms). The problem sets are labeled 10M-4R, 20M-20R, 50M-20R, 100M-20R, 200M-20R, 50M-50R, 100M-50R, 200M-50R, and 500M-50R, where xM-yR consists of problem instances with x meetings and y rooms. Each set contains 800 problem instances, giving a total of 7200 instances, obtained as follows.

We start from a set of real data from 32,065 unique meetings in a USC library collected by Kwak [20]. Each meeting request in this original data set includes the request arrival time, start time, duration, specified room and number of attendees. We first derive a probability distribution on meeting start times from

this data set. To obtain a set of requests, we sample x meetings for this distribution. We then create different instances with that set of requests by varying the time flexibility, request-to-start time gap, and temperature flexibility of the requests. The time flexibility of a request m is its number $|K_m| \in \{1, 2, 4, 8, 32\}$ of permissible start time steps. The request-to-start time gap denotes the duration $\{10 \text{ min}, 1 \text{ h}, 4 \text{ h}, 24 \text{ h}\}$ between the request's arrival time $\boldsymbol{a}_m$ and its first possible start time step. The temperature flexibility indicates the level of tolerance for the room temperature deviation from the standard heating (21°C) and cooling (23°C) setpoints, and is one of three settings: low, medium, or high flexibility, with $\boldsymbol{p}_m = 0.99$ for all settings, $\boldsymbol{T}_m^u = 2$ (low), 3 (medium), 5 (high), and $\boldsymbol{\alpha}_m = 10$ (low), 20 (medium), 30 (high). Note that in the high setting, the deviation could be up to 5°C, which is equivalent to 28°C for 30 min. This is an extreme case used to study the effects of temperature flexibility, but not a recommended setting. In the more realistic medium setting, the deviation is only up to 3°C, which is equivalent to 26°C for 20 min.

We keep the meeting duration and number of attendees identical to that of the original meeting request from the USC data, and assume that the occupant is fully flexible in terms of location, that is, that the meeting can be allocated to any room. In all problem sets, the duration $\mathbf{d}_m$ of meetings ranges from 1 to 4 time steps (30 min to 2 h). The meetings must be scheduled over a period of 5 summer days. The available rooms are located in 5 buildings and differ by their thermal resistance and capacitance [22]. We use a 1×4 zone layout where each zone has the same thermal resistance and capacitance as its neighboring zones. Moreover, all rooms have the same geometric area of $6 \times 10 \times 3$ m³ with a window surface area of 4×2 m² and a capacity of 30 people. The solar gain ranges from 50 to 350 W/m² during the day. All activities have between 2 and 30 attendees. All our experiments were run on a cluster consisting of a $2 \times$ AMD 6-Core Opteron 4334, 3.1 GHz with 64 GB memory.

5.2   Solution Method

To solve these problem instances, we combine our MIP model with Large Neighborhood Search as explained in [22]. LNS is a local search metaheuristic which iteratively improves an initial solution by alternating between a destroy and a repair step [30]. In brief, our LNS approach works as follows.

In every online session i, we start by generating an initial feasible solution, in two steps. First, we find a feasible occupancy schedule that minimizes the number of rooms used. Second, we determine the HVAC control settings (supply air flow rate and temperature) that minimize energy consumption for this schedule.

Our destroy step destroys part of the schedule by unscheduling the subset of new requests $N(i)$ that are allocated to two to four randomly selected locations. This forms an energy-aware meeting scheduling subproblem that is much smaller than the original problem and can be solved effectively using MIP. The repair step consists in repairing the schedule and re-optimizing the entire HVAC control by solving this subproblem using our MIP model. If this leads to an improved solution, then the new schedule and control settings are accepted. Otherwise,

we keep the solution that was just destroyed. Given that the LNS starts with a feasible solution and does not accept infeasible solutions, the solution remains feasible throughout the execution of the algorithm.

5.3   Online Vs. Offline Scheduling

We start by comparing the solution quality of our online approach with that of the offline approach [22]. In the online approach, the scheduler runs LNS for 5 min in each session, with a MIP runtime limit of 8 s in each iteration. In the offline approach, the entire set of requests to schedule is given, and we compute the final schedule; The scheduler runs LNS for 2 h, with a MIP runtime limit of 15 min in each iteration. To identify how much more improvement can be obtained, we warm start the offline schedule with the best online solution found (over all the possible request-to-start time gaps).

The difference of solution quality, that is the excess consumption of the online scheduling as a percentage of the off-line scheduling consumption, is shown in Fig. 4. The results for the fixed temperature setpoints are shown on the left, whilst those for the adaptive temperature setpoints are on the right. Both graphs show that the offline solutions are merely 1 % to 1.5 % better than the online solutions for tightly constrained problems (such as 200M-20R, 500M-50R), and that, as expected, the online approach improves when the problem is less constrained in terms of meetings to rooms ratio and temperature flexibility. Note that in the online approach, at most 20 requests arrive in each online session and a maximum of 4 rooms are destroyed, thus the sub-problems formed are small enough for MIP to solve them to (near) optimality. The off-line approach has many more meetings to deal with, but on the other hand, as problems become more constrained, it has more room to optimize than the greedy on-line approach. Altogether, even with a simple greedy approach, our online algorithm is able to perform effectively without prior knowledge of future requests.
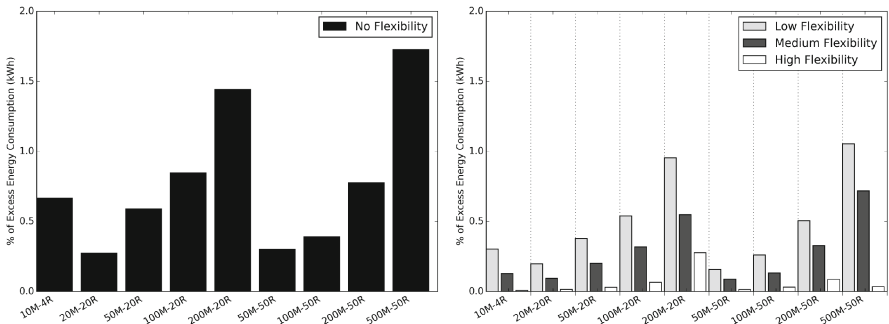
Fig. 4. Online vs. Offline Scheduling: With Fixed Temperature Setpoints (left) and Adaptive Temperature Setpoints (right)

5.4   Energy Savings of Adaptive Temperature Control

Next, we examine the benefits of our adaptive temperature control, which allows the occupant to specify their level of tolerance for the room temperature deviation from the fixed 21°-23°C comfort bounds. Because HVAC consumption is highly dependent on the temperature gap between the outdoor temperature and the occupied temperature setpoint, we show that even a small variation from the original setpoints can lead to large energy savings.

Figure 5 shows the additional energy savings obtained with adaptive temperature control as a percentage of the fixed temperature control consumption (left), and the maximum temperature deviation incurred by the adaptive approach (right). The left figure shows that the additional savings can reach up to [8 %, 12.7 %, 16.5 %] depending on the [low, medium, high] temperature flexibility allowed by the occupants. The right figure shows that the maximum degree of temperature deviation is only about [0.7, 1.5, 2.3]°C for low-to-high temperature flexibility, respectively. Overall, increasing temperature flexibility reduces HVAC consumption and cost. Taking an energy rate of $0.24/kWh and the 500M-50R problem set as example, this corresponds to annual savings of about [$11500, $19542, $24690] for [low, medium, high] temperature flexibility.
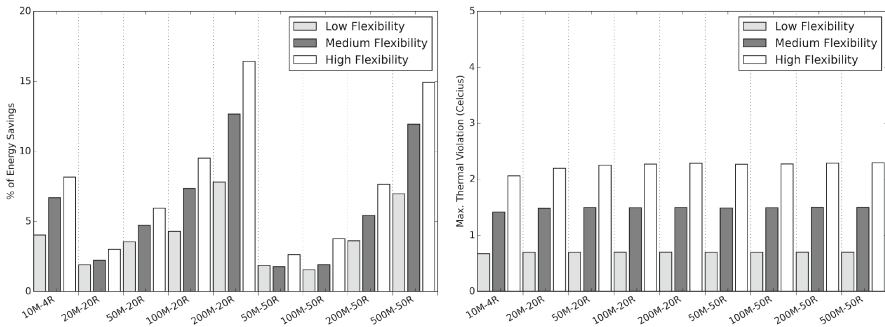
Fig. 5. Energy Savings from Adaptive Temperature Control (left) and Maximum Temperature Deviation from Standard Setpoints $(21° - 23°C)$

5.5   Model Feasibility

Finally, we study the solution feasibility of on-line scheduling with fixed and adaptive temperature control, respectively. Figure 6 shows the percentage of feasible solutions generated by the two approaches, as a function of the request-to-start time gap. Altogether, adaptive temperature control solves 73 % of the instances that are deemed unsolvable under the fixed temperature control regime.

We observed that with fixed temperature setpoints, we fail to generate feasible solutions in most cases when the requests arrive less than 1 h prior to the
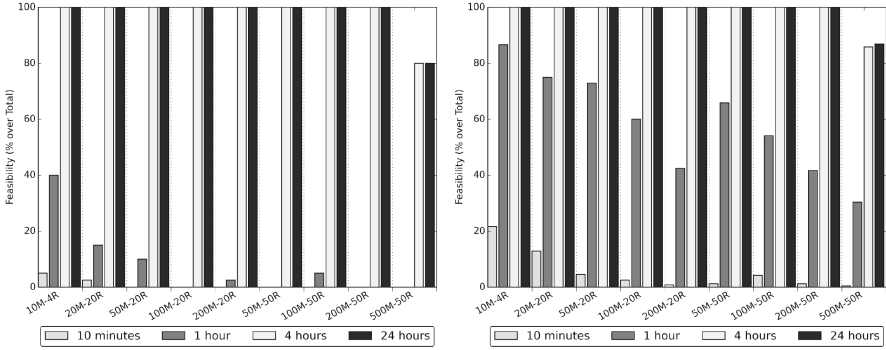
Fig. 6. Solution Feasibility: With Fixed Temperature Setpoints (left) and Adaptive Temperature Setpoints (right)

earliest possible activity start time. This infeasibility issue mainly happens at the initialization stage, where the initial schedule generation is decoupled from the initial HVAC control generation. In order to quickly generate an initial feasible schedule, activities are packed into the minimum number of rooms possible. However, the room temperatures may be too far from the temperature setpoints to obtain an initial feasible HVAC control reaching the designated occupied temperature at short notice. In contrast, the model with adaptive temperature control is able to solve many of these problem instances, and even generates some feasible solutions when the requests arrive just 10 min prior to the earliest activity start time. This is mainly due to the relaxation of the temperature setpoints. We observed that the number of feasible solutions increases proportionally to the temperature flexibility.

Apart from the constrainedness imposed on temperature setpoints, the model also stumbles into infeasibility when the scheduler fails to schedule all requests due to the lack of feasible location or time slot. Overall, the performance improves as the request-to-start time gap increases for both models.

6   Conclusions and Future Work

In this paper we develop an online scheduling model and adaptive temperature control method for joint HVAC control and occupancy scheduling. Leveraging an explicit model of building occupancy-based HVAC control, our model adopts a greedy approach to schedule dynamically arriving requests to take place at locations and times that are favorable from energy standpoint. Our experiments show that, even without prior knowledge of future requests, our model is able to produce energy-efficient schedules which are less than 1 % away from the clairvoyant solution.

We extend the model to enable adaptive temperature control, moving away from the conventional fixed comfort temperature setting. The occupant is allowed

to indicate their level of tolerance for the room temperature to deviate from the standard heating and cooling setpoints. We shows that thermal comfort flexibility significantly impacts energy consumption. Compared to the existing fixed temperature control, the energy savings in our experiments can reach up to 8 % with low temperature flexibility, with a maximum deviation of 0.7°C from the original setpoints, and up to 15 % with high temperature flexibility with a maximum of 2.3°C deviation from the standard setpoints. We have also shown that given some thermal comfort flexibility, our model is able to schedule requests arriving 10 min prior to the start time, and produce substantially more feasible solutions than the conventional fixed temperature setpoints approach.

We are interested in exploring new algorithmic approaches that allows us to improve our solution and scale even further. We are particularly interested in investigating stochastic scheduling and control, which allows us to predict future request arrival and cancellations. We are also interested in exploring the CP formulation of joint HVAC control and meeting scheduling. As the joint model consists of hybrid discrete-continuous variables, we plan to reformulate it by discretizing the HVAC control variables, and compare the solution quality generated by both MIP and CP models.

References

1. Aileen, E.: The potential energy savings through the use of adaptive comfort cooling setpoints in fully air conditioned australian office buildings, a simulation study. Equilibr. J. (2010)
2. Babonneau, F., Vial, J.P., Apparigliato, R.: Uncertainty and Environmental Decision Making. International Series in Operations Research and Management Science. Springer, Heidelberg (2009)
3. Ben-Tal, A., Nemirovski, A.: Robust convex optimization. Math. Oper. Res. **23**(4), 769–805 (1998)
4. Ben-Tal, A., Nemirovski, A.: Robust solutions to uncertain linear programs. OR Lett. **25**, 1–13 (1999)
5. Chai, B., Costa, A., Ahipasaoglu, S.D., Huang, S., Yuen, C., Yang, Z.: Minimizing commercial building cost in smart grid: an optimal meeting scheduling approach. In: 2014 IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 764–769. IEEE (2014)
6. Chew, B., Kazi, S., Amiri, A.: Adaptive thermal comfort model for air-conditioned lecture halls in Malaysia. World Acad. Sci. Eng. Technol. Int. J. Civ. Environ. Struct. Constr. Archit. Eng. **9**(2), 150–157 (2015)
7. De Dear, R.J., Brager, G.S., Reardon, J., Nicol, F., et al.: Developing an adaptive model of thermal comfort and preference/discussion. ASHRAE Trans. **104**, 145 (1998)

8. Dupont, C., Giuliani, G., Hermenier, F., Schulze, T., Somov, A.: An energy aware framework for virtual machine placement in cloud federated data centres. In: 2012 Third International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), pp. 1–10. IEEE (2012)

9. EIA: Us eia-department of energy, cbecs detailed tables (2003). http://www.eia.gov/consumption/commercial/

10. El Ghaoui, L., Lebret, H.: Robust solutions to least-squares problems with uncertain data. SIAM J. Matrix Anal. Appl. **18**, 1035–1064 (1997)

11. Department of Energy: Buildings energy data book. In: Buildings Energy Data Book. Department of Energy, United States (2011). http://buildingsdatabook.eren.doe.gov/ChapterIntro1.aspx

12. Gouda, M., Danaher, S., Underwood, C.: Low-order model for the simulation of a building and its heating system. Build. Serv. Eng. Res. Technol. **21**(3), 199–208 (2000)

13. Gouda, M., Danaher, S., Underwood, C.: Building thermal model reduction using nonlinear constrained optimization. Build. Environ. **37**(12), 1255–1265 (2002)

14. Goyal, S., Barooah, P.: A method for model-reduction of non-linear thermal dynamics of multi-zone buildings. Energy Build. **47**, 332–340 (2012)

15. Goyal, S., Ingley, H.A., Barooah, P.: Occupancy-based zone-climate control for energy-efficient buildings: complexity vs. performance. Appl. Energy **106**, 209–221 (2013)

16. Hijazi, H., Bonami, P., Ouorou, A.: Robust delay-constrained routing in telecommunications. Ann. Oper. Res. **206**(1), 163–181 (2013)

17. Ifrim, G., O'Sullivan, B., Simonis, H.: Properties of energy-price forecasts for scheduling. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 957–972. Springer, Heidelberg (2012)

18. Klein, L., Kwak, J.Y., Kavulya, G., Jazizadeh, F., Becerik-Gerber, B., Varakantham, P., Tambe, M.: Coordinating occupant behavior for building energy and comfort management using multi-agent systems. Autom. Constr. **22**, 525–536 (2012)

19. Kwak, J.y., Kar, D., Haskell, W., Varakantham, P., Tambe, M.: Building thinc: user incentivization and meeting rescheduling for energy savings. In: Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems, pp. 925–932 (2014)

20. Kwak, J.y., Varakantham, P., Maheswaran, R., Chang, Y.H., Tambe, M., Becerik-Gerber, B., Wood, W.: Tesla: An energy-saving agent that leverages schedule flexibility. In: Proceedings of the 12th International Conference on Autonomous Agents and Multi-agent Systems, pp. 965–972 (2013)

21. Lim, B.P., Van Den Briel, M., Thiébaux, S., Backhaus, S., Bent, R.: Hvac-aware occupancy scheduling. In: AAAI, pp. 4249–4250 (2015)

22. Lim, B.P., van den Briel, M., Thiébaux, S., Bent, R., Backhaus, S.: Large neighborhood search for energy aware meeting scheduling in smart buildings. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 240–254. Springer, Heidelberg (2015)

23. Majumdar, A., Zhang, Z., Albonesi, D.: Characterizing the benefits and limitations of smart building meeting room scheduling. In: Proceedings of the 7th International Conference on Cyber-Physical Systems (2016)

24. Majumdar, A., Albonesi, D.H., Bose, P.: Energy-aware meeting scheduling algorithms for smart buildings. In: Proceedings of the 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, pp. 161–168. ACM (2012)

25. Ono, M., Graybill, W., Williams, B.C.: Risk-sensitive plan execution for connected sustainable home. In: Proceedings of the 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, pp. 45–52. ACM (2012)
26. Pan, D., Wang, D., Cao, J., Peng, Y., Peng, X.: Minimizing building electricity costs in a dynamic power market: algorithms and impact on energy conservation. In: 2013 IEEE 34th Real-Time Systems Symposium (RTSS), pp. 107–117. IEEE (2013)
27. Pan, D., Yuan, Y., Wang, D., Xu, X., Peng, Y., Peng, X., Wan, P.J.: Thermal inertia: towards an energy conservation room management system. In: Proceedings of the 31st IEEE International Conference on Computer Communications, pp. 2606–2610. IEEE (2012)
28. Pitt, S. (ed.): Baseline Energy Consumption and Greenhouse Gas Emissions in Commercial Buildings in Australia Part 1 Report. Department of Climate Change and Energy Efficiency, Australia (2012). http://www.industry.gov.au/Energy/EnergyEfficiency/Non-residentialBuildings/Documents/CBBS-Part-1.pdf
29. Scott, P., Thiébaux, S., van den Briel, M., Van Hentenryck, P.: Residential demand response under uncertainty. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 645–660. Springer, Heidelberg (2013)
30. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
31. Ward, J., Wall, J., White, S.: Automate and motivate: behaviour-reliant building technology solutions for reducing greenhouse gas emissions. Archit. Sci. Rev. **53**(1), 87–94 (2010)
32. Yang, R., Wang, L.: Development of multi-agent system for building energy and comfort management based on occupant behaviors. Energy Build. **56**, 1–7 (2013)

Behavior Identification in Two-Stage Games for Incentivizing Citizen Science Exploration

Yexiang Xue[1(✉)], Ian Davies[2], Daniel Fink[2], Christopher Wood[2], and Carla P. Gomes[1]

[1] Computer Science Department, Cornell University, Ithaca, USA
{yexiang,gomes}@cs.cornell.edu
[2] Cornell Lab of Ornithology, Ithaca, USA
{id99,daniel.fink,chris.wood}@cornell.edu

Abstract. We consider two-stage games in which a leader seeks to direct the activities of independent agents by offering incentives. A good leader's strategy requires an understanding of the agents' utilities and the ability to predict agent behavior. Moreover, the optimization of outcomes requires an agent behavior model that can be efficiently incorporated into the leader's model. Here we address the agent behavior modeling problem and show how it can be used to reduce bias in a challenging citizen science application. Adapting ideas from Discrete Choice Modeling in behavioral economics, we develop a probabilistic behavioral model that takes into account variable patterns of human behavior and suboptimal actions. By modeling deviations from baseline behavior we are able to accurately predict future behavior based on limited, sparse data. We provide a novel scheme to fold the agent model into a bi-level optimization as a single Mixed Integer Program, and scale up our approach by adding redundant constraints, based on novel insights of an easy-hard-easy phase transition phenomenon. We apply our methodology to a game called Avicaching, in collaboration with eBird, a well-established citizen science program that collects bird observations for conservation. Field results show that our behavioral model performs well and that the incentives are remarkably effective at steering citizen scientists' efforts to reduce bias by exploring under-sampled areas. Moreover, the data collected from Avicaching improves the performance of species distribution models.

1 Introduction

Many game applications involve a leader, who commits to a strategy before her followers. Thus in order to come up with an optimal strategy, the leader must factor in the reasoning process of her followers. This leads naturally to the following bi-level optimization:

$$\textbf{Leader: } \underset{\mathbf{a}_1}{\text{maximize}} \quad U_L(\mathbf{a}_1, \mathbf{a}_2),$$

$$\text{subject to} \quad \textbf{Followers: } \mathbf{a}_2 \leftarrow \underset{\mathbf{a}_2}{\arg\max} \, U_F(\mathbf{a}_2, \mathbf{a}_1).$$

Here the leader's utility function U_L is known a priori, but the utilities of the individual followers U_F are unknown by the leader. $\mathbf{a}_1$ and $\mathbf{a}_2$ are the actions of the leader and the followers, respectively.

At the heart of solving this problem lies the challenge of identifying the utility functions that govern the followers' behavior. On one hand, the behavioral model has to be capable of capturing complex, highly variable human behavior and it should be robust to make predictions with limited, sparse data. On the other hand, the behavioral model has to be efficiently incorporated into the overall bi-level optimization problem.

In this paper, we address the behavioral identification problem in two-stage games to reduce data bias in citizen science projects, such as *Zooniverse*, *Coralwatch*, and *eBird* [7,20,29]. These projects use crowdsourcing techniques to engage the public as agents in the data collection process to address scientific questions determined by project leaders. Despite their tremendous success, the data collected often suffer from biases, which arises from fundamental mismatches between the personal motivations that determine how individual agents collect data and the data needs for scientific inquiry. For example, projects that allow participants to choose where and when to make observations tend to collect the most data near areas of human activity (see Fig. 1). Uneven geographic (and temporal) data density presents a challenge for scientific studies.
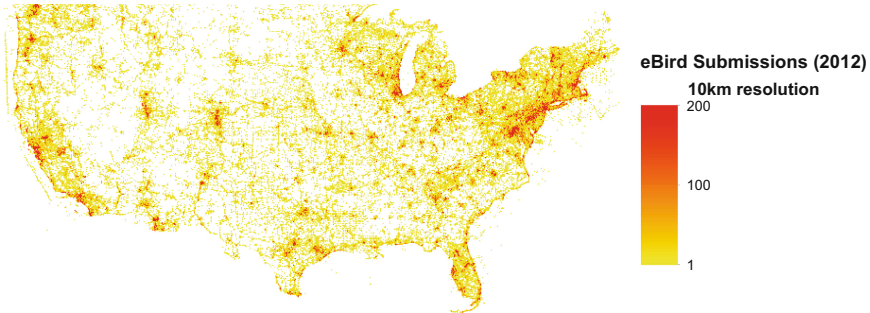

Fig. 1. Number of observations submitted to *eBird* in 2012 in the Continental US. Submissions are biased towards population centers.

Previous work has shown that games are effective in steering citizen scientists towards crucial scientific tasks [31]. Under a two-stage game scenario, individual participants are offered incentives to spend more effort collecting data at sites identified as important by project leaders. In this gamified setting, a key problem is the *optimal reward allocation* problem: how to design a reward scheme which maximizes citizen scientists' overall contribution to science.

The reward allocation problem is closely related to the Principal-Agent Problem, first raised in behavioral economics [26]. More recently, it has also been studied in computer science [1,3,10,14,15]. It is also related to the Stackelberg pricing games [8,9,11,22], in which the leader commits to a strategy before her

followers. In crowdsourcing, related work includes mechanisms to improve the crowd performance [2, 4, 6, 17, 19, 23, 27, 28, 30]. The reward allocation problem is a bi-level optimization that includes as a crucial component the modeling of citizen scientists' behavior.

Here we propose **a novel probabilistic model to capture agents' behavior in two-stage games**, adapting ideas from *Discrete Choice Modeling* in behavior economics [21], as well as **a novel Mixed Integer Programming encoding to solve the reward allocation problem,** in which the proposed probabilistic behavioral model is folded as linear constraints. We also **apply our novel behavioral model into a real citizen science domain**. Our contributions are multi-dimensional:

– On machine learning side, our proposed behavioral model is **(1)** *structural*, meaning that its parameters provide intuitive insights into agents' decision-making process, as well as **(2)** *generative*, meaning that it can generalize to new circumstances with different environmental features and reward treatments. Unlike the knapsack model in previous work [31], our model is **(3)** *probabilistic*. Therefore it is able to account for complex human behavior, as well as suboptimal actions. Instead of directly modeling agents' preferences, which would be difficult to capture, we break the model into **(4)** a *conditional* form, and focus on modeling agents' *deviation* from their baseline behavior under zero reward treatments, alleviating the data sparsity problem by effectively *taking advantage of the relatively abundant historical data before the introduction of the reward game.*
– On the inference side, despite the fact that the reward allocation problem is a bi-level optimization, we are able to **(5)** *fold the behavioral model into the global problem as a set of linear constraints*, therefore the entire reward allocation problem is solved with a *single Mixed Integer Program (MIP)*. In addition, we add redundant constraints to trigger pruning, therefore scaling up the MIP encoding to large instances, based on observations of a novel **(6)** *easy-hard-easy phase transition phenomenon* [13] in the empirical complexity.
– On the application side, we **(7)** apply our behavioral model into a recently launched gamification application called *Avicaching* [31], in the well-established *eBird* citizen science program. Our behavioral model is able to better capture the decision process of the participants than previously proposed models with real field data. Furthermore, the reward designed by the optimal reward allocation algorithm proves to be effective in minimizing the bias in *eBird* data collection process.
– Finally, in terms of addressing *the core scientific goal of ebird*, we show the benefit of having data collected from the *Avicaching* game by demonstrating **(8)** a clear boost in the performance of species distribution modeling when adding data from Avicaching locations.

2   Two Stage Game for Bias Reduction

In our two-stage game setting, citizen scientists visit a set of locations and report their observations of events of interest in those locations. Our model can be generalized to other scientific exploration activities as well [31]. The incentive game involves two self-interested parties: the organizer and the agents. On one side, rational agents (e.g., citizen scientists) select a set of locations to visit that maximizes their own utilities under budgets. On the other side, the organizer (e.g., a citizen science program) uses rewards to encourage agents to visit locations with large scientific value. For example, in *eBird*, bird watchers choose their sites to visit based on a combination of environmental values, personal preference and convenience. The organizer in turn sets external rewards at different locations to promote uniform exploration activities. At a high level, this leads to a bi-level optimization problem:

$$\textbf{Organizer:} \underset{\mathbf{r}}{\text{maximize}} \quad U_o(\mathbf{v}, \mathbf{r}),$$
$$\text{subject to} \quad \textbf{Agents: } \mathbf{v} \leftarrow V_a(\mathbf{f}, \mathbf{r}). \tag{1}$$

In this formulation, $\mathbf{r}$ is the external reward that the organizer uses to steer the agents, and $\mathbf{v}$ are the response from the agents, affected by internal utilities, which is determined by feature vector $\mathbf{f}$, and external rewards $\mathbf{r}$ set by the organizer. $U_o(\mathbf{v}, \mathbf{r})$ is the utility function of the organizer, which depends on agents' response $\mathbf{v}$.

Addressing the Organizer-Agent Problem requires a good behavioral model for agents $V_a(\mathbf{f}, \mathbf{r})$, which involves challenges from two associated problems: one is the *Identification Problem* and the other one is the *Pricing Problem*. For the ***Identification*** Problem, we need to learn an agent model to predict noisy human behavior under different reward treatments. For the ***Pricing*** Problem, we need to incorporate the identified agent model into the bi-level optimization (shown in Eq. 1) to solve the overall reward allocation problem.

The organizer's goal is to promote a balanced exploration activity. Let $L = \{l_1, l_2, \ldots, l_n\}$ be the set of locations, and y_i be the amount of effort agents devote to location l_i. We normalize y_i so that $\sum_{i=1}^{n} y_i = 1$. In other words, y_i is proportional to the number of observations submitted at location l_i. Denote by $\mathbf{y}$ the column vector $(y_1, \ldots, y_n)^T$ and by $\overline{\mathbf{y}}$ the constant column vector $(\overline{y}, \ldots, \overline{y})^T$ where $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{1}{n}$. To promote a uniform sampling activity, we model the organizer's objective as to minimize the bias in agents' sampling effort: minimize $D_p = \frac{1}{n}||\mathbf{y} - \overline{\mathbf{y}}||_p^p$. Given this definition, D_1 corresponds to the *mean absolute deviation*, while D_2 corresponds to the *sample variance*. Other objectives could be used, e.g., maximizing the entropy of $\mathbf{y}$ in order to minimize its distance to a uniform distribution.[1]

---

[1] Uncertainty measures, often used in active learning [25], are typically tied to one particular predictive model. We did not use them because of the need to meet multiple scientific goals in our application.

3   Probabilistic Behavior Model

A key to solving the reward allocation problem is to identify a good behavioral model, which captures agents' preferences to environmental features as well as external rewards. It is challenging, given (1) the *complex and highly variable human behavior*, which cannot be fully captured by environmental variables. Moreover, (2) the data collected with an incentive game in the field is *limited*, since we cannot afford to alienate the community by changing the rewards dramatically. On the other hand, there is much historical data for participants collected without the reward game. How to make full use of this piece of data becomes an interesting question. (3) To efficiently *support decision making*, our behavioral model needs to be able to *fit nicely into the bi-level optimization framework of the pricing problem*. In this paper, we introduce a novel probabilistic model to capture the agents' behavior.

- It takes a **structural** approach, which jointly learns how agents distribute their effort among all locations, rather than predicting the amount of effort spent in each location independently.
- We adopt the idea of **the Discrete Choice Model** in behavioral economics [21], which captures agents' noisy behavior as well as suboptimal actions.
- We alleviate the data sparsity problem by focusing on modeling the conditional probabilities characterizing people's **deviation** from their normal behaviors under no reward treatments, thus effectively taking advantage of relatively abundant historical data without rewards.
- Finally, this structural and generative model allows us to *fold the agents' model as a set of linear constraints* into the reward allocation problem, therefore the entire problem can be solved by **a single MIP**.

During one round of reward treatment, suppose we offer an agent an extra reward r_i for one observation made at location i. Let $\mathbf{r} = (r_1, \ldots, r_n)^T$ be the reward vector. Let $y_{j,i}$ be the amount of effort that agent j devote to location i. We normalize the effort such that $\sum_{i=1}^{n} y_{j,i} = 1$. Let $\mathbf{y}_j = (y_{j,1}, \ldots, y_{j,n})^T$ be vector characterizing the distribution of effort.

Behavioral modeling is to fit a function $\mathbf{y}_j = V_a(\mathbf{f}, \mathbf{r})$ which predicts how agent j distributes his effort $\mathbf{y}_j$ based on environmental features $\mathbf{f}$ and the current reward vector $\mathbf{r}$. One option is to fit V_a as a joint distribution. Unfortunately, this is challenging given the multitude of subtle factors affecting human behavior. Luckily, most participants in our reward game participate heavily in eBird. We have much historical data on them before the reward game, so we hope to use this data to capture their subtle preferences. We therefore break down the agents' behavior into a conditional form, comprising each participant's historical preferences $\mathbf{x}_j$ without external rewards, and the deviation of new behavior $\mathbf{y}_j$ under reward treatment from the baseline behavior $\mathbf{x}_j$. $\mathbf{x}_j$ is summarized based on agents' past behavior during the same time of the year, across previous years. For recently joined participants, we use the population mean as their baseline distribution.

We focus on modeling the *conditional* part, which predicts the *deviation* of people's behavior from $\mathbf{x}_j$ to $\mathbf{y}_j$. Notice that it is a simpler problem than fitting V_a as a joint distribution directly, because *the only main effect that is in the field during the reward treatment period of $\mathbf{y}_j$, but not in the baseline treatment period of $\mathbf{x}_j$, is the introduction of reward* $\mathbf{r}$. Therefore, the effects of rewards are much stronger in the conditional distribution. We model the transformation matrix P connecting $\mathbf{y}_j$ and $\mathbf{x}_j$, which depends on internal utility features $\mathbf{f}$, and external rewards $\mathbf{r}$:

$$\mathbf{y}_j = P(\mathbf{f}, \mathbf{r})\, \mathbf{x}_j. \tag{2}$$

Many machine learning applications share similar ideas as ours in terms of modeling the conditional part in the joint data distribution [12,18]. Let $p_{u,v}$ be the entry of matrix P at the u-th row and the v-th column. Intuitively, $p_{u,v}$ denotes the proportion of effort that originally was spent in location v, but has been shifted to location u. Motivated by the Discrete Choice Model in behavioral economics [21], we further parameterize the matrix P as:

$$p_{u,v} = \frac{\exp(\mathbf{w} \cdot \phi(\mathbf{f}_{u,v}, r_u))}{\sum_{u'} \exp(\mathbf{w} \cdot \phi(\mathbf{f}_{u',v}, r_{u'}))}. \tag{3}$$

In this formulation, $\mathbf{f}_{u,v}$ is the environmental feature vector for the transition from location v to location u, which includes features for location u and v individually, such as underlying landscapes, interesting species to see, historical popularities, as well as features that depend on both the two locations, such as the traveling distance, etc. ϕ is a function that maps features to a high dimensional space, which includes singular effect terms as well as cross effect terms. $\mathbf{w}$ is a vector that gives relative weights to different features in the output space of ϕ. The dimensionality of $\mathbf{w}$ is the same as the output of function ϕ.

In previous work [31], agents' behavior are modeled as solving knapsack problems: agents select the best set of locations to visit, which jointly maximizes the reward $\mathbf{w} \cdot \phi(\mathbf{f}_{u,v}, r_u)$, subject to a cost constraint. Since Eq. 3 is a softmax function, our proposed model can be viewed as an extension of the knapsack model to the probabilistic case. Indeed, suppose agents always take the optimal action (as in the knapsack case), their behavior will demonstrate a logit form as shown in Eq. 3, if apart from the features in $\phi(\mathbf{f}_{u,v}, r_u)$, their actions are further affected by a set of factors that are only known to agents themselves and with an extreme value distribution [24].

Nevertheless, compared to the knapsack model, our behavioral model is considerably more realistic. Our model is probabilistic, thus it is able to represent variability in agent behavior, as well as uncertainty on the part of the organizer. Suppose one agent chooses to visit either location A or B, but with $70\,\%$ chance for A, and $30\,\%$ chance for B. The deterministic knapsack model has to learn a utility function that either predicts that A is a better option than B or vice versa. Our model can come up with an optimal reward scheme in this probabilistic setting. Besides, in the knapsack model, agents' behavior is subject to a strict budget limit. In reality, people occasionally venture beyond their normal travel

distance. Our model is able to capture this aspect, by learning a soft penalty on the traveling distance.

3.1 Identification Problem

The identification problem learns the parameters of the agents' behavior model by examining agents' responses to various reward treatments. Specifically, we are given a dataset $\mathcal{D}$ composed of quadruples $(\mathbf{x}_{j,t}, \mathbf{y}_{j,t}, \mathbf{r}_t, \mathbf{f}_t)$, in which $\mathbf{x}_{j,t}$ and $\mathbf{y}_{j,t}$ are the visit densities of one citizen scientist without and with the reward treatment $\mathbf{r}_t$. $\mathbf{f}_t$ is the environmental feature vector during the period of the treatment. We need to identify weights $\mathbf{w}$ that best matches $\mathbf{y}_{j,t}$ with $P(\mathbf{f}_t, \mathbf{r}_t; \mathbf{w}) \, \mathbf{x}_{j,t}$. Using the L2 loss, we minimize the following empirical risk function:

$$R(\mathbf{w}) = \sum_{j,t} \left(u_{j,t}(\mathbf{y}_{j,t} - P(\mathbf{f}_t, \mathbf{r}_t; \mathbf{w}) \, \mathbf{x}_{j,t}) \right)^2. \tag{4}$$

Here, instances are weighted by $u_{j,t}$, which is the total number of submissions of the corresponding citizen scientist during one reward treatment r_t. We fit a common $\mathbf{w}$ for all citizen scientists, due to limited amount of data.

   We specify regularizers to prevent overfitting. It is a common practice to penalize the norm of $\mathbf{w}$ in regularizers. However, when the data is uninformative, a baseline model should always make predictions based on baseline density, i.e., predict $\mathbf{y} = \mathbf{x}$. This suggests that matrix P should be close to the identity matrix in such uninformative case. However, setting $\mathbf{w} = \mathbf{0}$ will make all $p_{u,v} = \frac{1}{n}$ according to Eq. 3, which renders P away from the identity matrix. In this case, we add an indicator variable $\mathbf{1}_{u,v}$ as a special feature. $\mathbf{1}_{u,v} = 1$ if and only if $u = v$, and the entries in matrix P becomes:

$$p_{u,v} = \frac{\exp(\mathbf{w} \cdot \phi(\mathbf{f}_{u,v}, r_u) + \eta \cdot \mathbf{1}_{u,v})}{\sum_{u'} \exp(\mathbf{w} \cdot \phi(\mathbf{f}_{u',v}, r_{u'}) + \eta \cdot \mathbf{1}_{u',v})}. \tag{5}$$

P now becomes close to an identity matrix if $\mathbf{w}$ is close to $\mathbf{0}$ and η is positive. We minimize the following augmented risk function:

$$R(\mathbf{w}) = \sum_{j,t} \left(u_{j,t}(\mathbf{y}_{j,t} - P(\mathbf{f}_t, \mathbf{r}_t; \mathbf{w}) \, \mathbf{x}_{j,t}) \right)^2 + \lambda \cdot |\mathbf{w}|_1. \tag{6}$$

Here, the classical L1 regularizer $\lambda \cdot |\mathbf{w}|_1$ helps identify important factors by learning a sparse $\mathbf{w}$ vector. Apart from tuning λ, we also tune η to control how closely the predicted $\mathbf{y}$ should match historical densities $\mathbf{x}$. The minimization problem in Eq. 6 can be solved by gradient descent. We use BFGS algorithm [5], which further accelerates descent using second order information.

3.2 Pricing Problem

Given a learned behavioral model, the pricing problem is to minimize the spatial bias D_p, subject to the behavioral model:

$$\begin{aligned}
\underset{\mathbf{r}}{\text{minimize }} D_p &= \frac{1}{n}||\mathbf{y} - \bar{\mathbf{y}}||_p^p, \\
\text{subject to } \mathbf{y} &= P(\mathbf{f}, \mathbf{r}; \mathbf{w}) \, \mathbf{x}, \\
r_i &\in R.
\end{aligned} \tag{7}$$

In this formulation, $\mathbf{x} = (x_1, \ldots, x_n)^T$ is the normalized distribution of effort among all agents. Matrix P is learned from the approach given in the previous section. In practice, because people are more accustomed to only a few distinct reward levels, we further restrict r_i to take a set of discrete values in set R.

The main challenge to solve the pricing problem is the sum-exponential form of the entires of matrix P (Eq. 5). Nevertheless, in this paper we are able to show that the sum-exponential form can be captured by a set of linear constraints. Therefore *the pricing problem can be formalized as a single Mixed Integer Program (MIP)*.

Suppose R has K different reward levels: $R = \{R_1, \ldots, R_K\}$. Introduce indicator variables $dr_{i,k}$ for $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, K\}$. $dr_{i,k} = 1$ if and only if r_i, the reward for location i, is R_k. r_i can take only one value in R, so $dr_{i,k}$ should satisfy:

$$\sum_{k=1}^{K} dr_{i,k} = 1, \quad \forall i \in \{1, \ldots, n\}. \tag{8}$$

The challenge is the sum-exponential operator in Eq. 5. To overcome this difficulty, we introduce extra variables α_v ($\alpha_v \geq 0$) for $v \in \{1, \ldots, n\}$, and we use linear constraints to enforce

$$\alpha_v = \frac{1}{Z_v} = \frac{1}{\sum_{u'} \exp(\mathbf{w} \cdot \phi(\mathbf{f}_{u',v}, r_{u'}) + \eta \cdot \mathbf{1}_{u',v})}. \tag{9}$$

Here Z_v is the partition function in Eq. 5. We first substitute α_v into Eq. 5, and get:

$$p_{u,v} = \exp(\mathbf{w} \cdot \phi(\mathbf{f}_{u,v}, r_u) + \eta \cdot \mathbf{1}_{u,v}) \cdot \alpha_v. \tag{10}$$

However, in this case both r_u and α_v are variables, so Eq. 10 is not linear. To linearize it, we rewrite this equation in the following conditional form:

$$\begin{aligned}
dr_{u,k} = 1 \implies p_{u,v} &= \alpha_v \exp(\mathbf{w} \cdot \phi(\mathbf{f}_{u,v}, R_k) + \eta \cdot \mathbf{1}_{u,v}), \\
&\forall k \in \{1 \ldots K\}, \forall u, v \in \{1 \ldots n\}.
\end{aligned} \tag{11}$$

Here, $\mathbf{w}$ is learned from the identification problem, so it is a constant in the pricing problem. When r_u is fixed to R_k ($dr_{u,k} = 1$), $\exp(\mathbf{w} \cdot \phi(\mathbf{f}_{u,v}, r_u) + \eta \cdot \mathbf{1}_{u,v})$ becomes a constant, so the right-hand side of Eq. 11 is indeed a linear equation over α_v. We can enforce the conditional constraints using the big-M formulation. Next, we require the columns of P sum to 1:

$$\sum_{u=1}^{n} p_{u,v} = 1, \quad \forall v \in \{1, \ldots, n\}. \tag{12}$$

It can be shown in the following Theorem that Eqs. 11 and 12 guarantee that $\alpha_v = 1/Z_v$. Further because of Eq. 10, we must have the fact that $p_{u,v}$ satisfies the sum-exponential form in Eq. 5.

Theorem 1. *Equations 11 and 12 guarantee that $\alpha_v = 1/Z_v$, $\forall v \in \{1, \ldots, n\}$.*

Proof. Equation 11 forces α_v to be proportional to $1/Z_v$ and Eq. 12 constrains the sum of $p_{u,v}$ to be 1.

Next we model the objective function D_p. Here we provide a formulation for D_1.[2] The key is to model the absolute difference $|y_i - \overline{y}|$. Introduce variable t_i for $|y_i - \overline{y}|$, $i \in \{1, \ldots, n\}$, and constraints $t_i \geq y_i - \overline{y}$ and $t_i \geq \overline{y} - y_i$ to guarantee that $t_i \geq |y_i - \overline{y}|$. Then we can modify the objective so as to minimize $\sum_{i=1}^{n} t_i$.

In practice, we find the MIP encoding with the constraints in Eqs. 8–12 does not scale well with small external rewards (see Sect. 4.3). In this case, we add redundant constraints to facilitate constraint propagation and pruning. When $r_u = R_k$, we add these redundant constraints:

$$p_{u,v} \leq \frac{\exp(g_{u,v}(R_k))}{\exp(g_{u,v}(R_k)) + \sum_{u' \neq u} \exp(\min_{r \in R} g_{u',v}(r))}, \tag{13}$$

and

$$p_{u,v} \geq \frac{\exp(g_{u,v}(R_k))}{\exp(g_{u,v}(R_k)) + \sum_{u' \neq u} \exp(\max_{r \in R} g_{u',v}(r))}. \tag{14}$$

Here, $g_{u,v}(r)$ is an abbreviation for $\mathbf{w} \cdot \phi(\mathbf{f}_{u,v}, r) + \eta \cdot \mathbf{1}_{u,v}$. The right hand side of these two inequalities are clearly the upper and lower bound of $p_{u,v}$, because all free variables are fixed to their most extreme values.

4   Experiments

4.1   Applying the Behavioral Model to *Avicaching*

We apply our behavioral model into ***Avicaching*** [31], a recently launched gamified application to reduce the data bias problem within *eBird*, a well-established citizen science program. *Avicaching* is created in the spirit of promoting "friendly competition and cooperation" among *eBird* participants. *Avicaching* started in

---

[2] One needs solve a Mixed Quadratic Program if he uses objective function D_2.

March 2015 as a pilot study in Tompkins and Cortland counties, New York. A set of publicly accessible locations with no prior eBird observations were defined as *Avicaching* locations: bird watchers received extra *avicaching* points for every checklist they submitted in those locations. These locations were selected around under-covered regions from the current *eBird* dataset, emphasizing important yet under-sampled land types, such as agricultural land and forest. *Avicaching* points have intrinsic value to bird watchers, because they mark their scientific contribution to *eBird*. In addition, other rewards, such as binoculars, were also provided in the form of a lottery, which is based on the total *avicaching* points earned by each participant. The *Avicaching* points were updated every week. The probabilistic behavioral model was used in the bi-level optimization problem, which allocates optimal rewards to locations to minimize the spatial bias. We used the participants' response in the first few weeks to train our behavioral model.
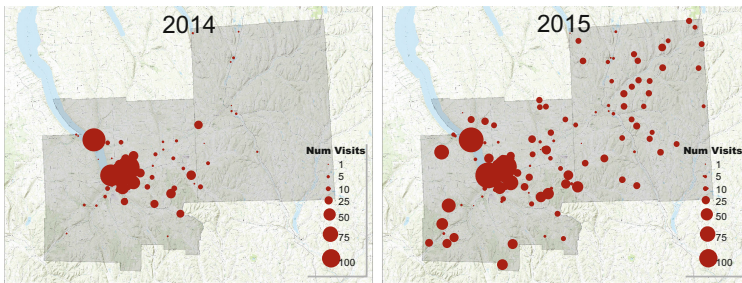
Fig. 2. The comparison of the locations of submissions in *eBird* in Tompkins and Cortland County in New York State. The size of the circles represent the number of submissions. (Left) from Mar 28 to Aug 31, 2014, before *Avicaching*. (Right) from Mar 28 to Aug 31, 2015, after *Avicaching* is introduced. Effort is shifted towards under-sampled locations significantly. Study area is shaded.

Encouraged by *Avicaching*, bird watchers shifted their effort towards under-sampled locations. As visually demonstrated in Fig. 2, 482 *eBird* observations were submitted from *Avicaching* locations, out of the 2,522 observations in total for Tompkins and Cortland County during summer months from June 15 to Sep 15, 2015. 19.1 % of birding effort has shifted from oversampled locations to under-sampled *Avicaching* locations, which received zero submissions before. Cortland, as an under-sampled county, received 202 observations during these three summer months in 2015, when *Avicaching* is in the field, which is 2.3 times the number of visits of the previous two years combined (there are in total 87 submissions from Cortland during the same period of time in 2013 and 2014). In terms of uniformity, the normalized D_2 score ($\frac{1}{n}||\mathbf{Y} - \overline{\mathbf{Y}}||_2^2/\overline{Y}$), dropped from 0.017 in 2013, 0.017 in 2014 to 0.013 in 2015 during the period of time.

4.2 Evaluation of the Probabilistic Behavioral Model for the Identification Problem

The behavioral model used in the reward allocation problem of one week is fit using the data since the beginning of *Avicaching* and up to that week. The data are composed of $(\mathbf{x}_{j,t}, \mathbf{y}_{j,t}, \mathbf{r}_t, \mathbf{f}_t)$ tuples, each of which represents the density of locations a bird watcher visited during one week's reward treatment. There are in total 116 locations in total in this two counties (the length of $\mathbf{x}_{j,t}$ and $\mathbf{y}_{j,t}$), out of which 50 are *Avicaching* locations. We split the dataset into 75 % for training, 5 % for validation, and the remaining 20 % for testing. The data for validation is used to select the values of regularizers. We found the model is not sensitive to the values of regularizers, as long as they are in a proper range. The reported performance is averaged over 3 random splits. The location features we consider for the behavioral model are: the number of visits in each month (popularity), the expected number of species to see (interestingness), the NLCD covariates for the landscape [16], housing density (population center), elevation, distances to rivers, roads, etc., latitude and longitude (geographical regions), convenience factor (distance to reach), and *Avicaching* points (rewards). We also include non-linear transformation of these features and cross terms.

We compare our proposed model with three baseline models. The first model always uses historical density to make predictions, i.e., always predict $\mathbf{y}_{j,t} = \mathbf{x}_{j,t}$. The second model is the structural SVM model from [31], a powerful nonparametric machine learning model optimized for solving knapsack problems. The third benchmark is a continuous-response random forest, which predicts the density $y_{j,t}$ at each location independently with 1,000 trees of depth 10. Random forests are
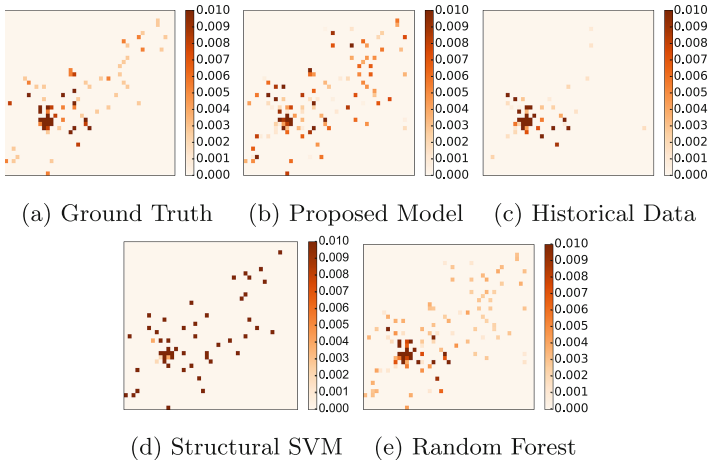

(a) Ground Truth   (b) Proposed Model   (c) Historical Data

(d) Structural SVM   (e) Random Forest

Fig. 3. The comparison of probabilities of visiting each location predicted by various behavioral model on one test set. The range was selected to highlight locations with small probabilities. The proposed model matches closest to the ground truth (note the color scale). (Color figure online)

Table 1. Comparison of predicted performance on the test set. The table shows the normalized mean squared error (MSE). Our proposed model outperforms the other 3 baseline models.

Method	Normalized MSE
Proposed	**0.26**
Historical	0.36
Structural SVM	0.93
Random Forest	0.37

expected to set the benchmark for very good predictive performance. However, the lack of interpretable structures precludes them from being folded into the MIP formulation of the pricing problem. Both the Structural SVM and the random forest model share the same environmental features as our proposed model. We include the baseline density $\mathbf{x}_{j,t}$ in the two models as a separate feature.

Table 1 shows the comparison on normalized mean squared error (MSE), which is $\frac{\sum_{j,t}\sum_{i=1}^{n}(u_{j,t}(y_{j,t,i}^{truth}-y_{j,t,i}^{pred}))^2}{\sum_{j,t}\sum_{i=1}^{n}(u_{j,t}(y_{j,t,i}^{truth}-\overline{y_{j,t,i}^{truth}}))^2}$. Here $y_{j,t,i}^{truth}$ is the true density for agent j in time t, and $y_{j,t,i}^{pred}$ is the predicted value at location i. The squared error is further weighted by $u_{j,t}$ – the number of submissions during the reward period.

Our proposed model clearly outperforms the other 3 models. To further visualize the difference, the predicted probabilities to visit each location, averaged over all test cases in one test set, are compared with the ground truth in Fig. 3. The locations with high probabilities (shown with dark red cells) are historically popular sites. The model based on historical density predicted very well on these sites, because we have rich data on people's birding history, and bird watchers' behavior is relatively stable across different years. Those sites with relative low probabilities (light orange cells) are often under-sampled sites with *Avicaching* rewards. In this case, the historical model missed completely. Structural SVM performed the worst. While random forest performed well qualitatively, it was out-performed by our proposed model (Table 1). Even if the random forest model had comparable performance, it cannot be folded into the MIP to solve the bi-level optimization problem.

4.3   Phase Transition on the Pricing Problem

The scalability of the Mixed Integer Programming encoding proposed for the pricing problem is also important. To evaluate the solver, we generated 5 sets of synthetic instances, with numbers of locations n ranging from 15 to 35. Each set had 30 instances with the same n, generated in a way to best mimic people's behavior. To make it easy for plotting, reward set R contains 2 levels of rewards for these instances: one was 0, and the other was a non-zero reward shown in the horizontal axis of Fig. 4 (all 30 instances in one test set shared a common non-zero reward). We kept all other parameters the same, and only varied the

non-zero rewards. The curves in Fig. 4 report the median time to solve these instances with MIP encoded in CPLEX 12.6, with a single Intel x5690 core and 8 GB of memory. Each dot in one curve represents the median time of solving 30 instances in one test set. Two points on a given curve only differ in the reward level.
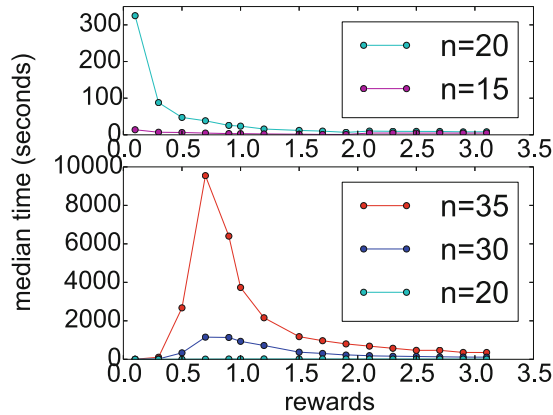

Fig. 4. The easy-hard-easy phase transition for the pricing problem; n is the number of locations. (Upper) The median time to solve instances with various non-zero rewards without the redundant constraints in Eqs. 13 and 14. The time is long for instances with small rewards. (Lower) The median time when redundant constraints are introduced. The easy-hard-easy pattern emerges.

Intuitively, there should be an easy-hard-easy pattern in the empirical complexity of the pricing problem. If the external rewards are too small, then it makes little difference in terms of changing agents' behavior whether one reward is assigned to one location or not. On the other hand, if the rewards are too large, then agents' behavior is completely dominated by these external rewards. It is when the external rewards match agents' internal utilities that the problem becomes hard, and the algorithm needs to plan wisely in allocating rewards. Nevertheless, when the redundant constraints in Eqs. 13 and 14 were not introduced (Upper Panel of Fig. 4), we did not see the easy-hard-easy pattern. Problem instances with small non-zero external rewards were significantly harder than other ones.

The unexpected long runtimes for instances with small rewards were due to the difficulty in propagating constraints. The solver could not automatically discover the fact that the reward was too small to have any substantial impact, so it spends much time on many meaningless branches. This prevented the solver from early pruning, which was often the key to efficient problem solving. Noticing this aspect, we added redundant constraints (Eqs. 13 and 14) into the MIP formulation. These two equations were obvious necessary conditions for $p_{u,v}$. Adding these two equations helped the solver find bounds on $p_{u,v}$, so it could prove tighter bounds for the objective function, and trigger early pruning more often.

After adding these two constraints, the easy-hard-easy phenomenon emerged. We were also able to scale up to larger instances due to better constraint propagation (It takes too long for the solver to run for $n = 30$ and $n = 35$ without additional constraints, so they are not plotted in the upper panel of Fig. 4).

4.4   Benefit of Avicaching on Species Modeling

We are able to see the benefit of having data collected from avicaching locations on species distribution modeling – the main scientific application of *eBird* data. To fit the species distribution models, we use the data from April to June (the spring migration period), in both Tompkins and Cortland counties, including avicaching and non-avicaching locations. We predict the occupancy of a species based on environmental variables. For each species, we fit random forest models with 1,000 trees, with each tree at the depth of 10.

(a) Ground Truth   (b) with Avicache   (c) without Avicache   (d) bird
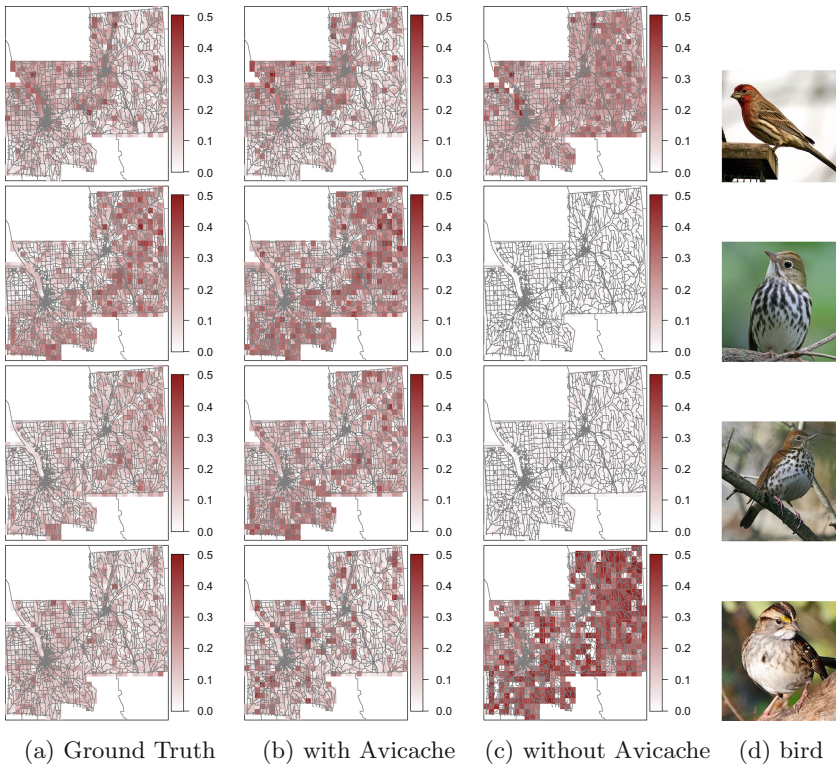
Fig. 5. The benefit of having observations from avicaching sites. (1st Row) Model for House Finch; (2nd Row) Ovenbird; (3rd Row) Wood Thrush; (4th Row) White-throated Sparrow. Predictive model fit with 2015 data including that from Avicaching sites in Cortland (2rd column) better matches a model close to the ground truth (1st column, fit with all available data, best effort and validated by experts), compared with the model fit without Avicaching data (3rd column).

Figure 5 shows the predicted probabilities of occurrence in heatmaps for random forest models fit with different datasets, for four species in the two counties. The first column shows the distribution models fit with the most comprehensive dataset, which consists of data from both counties, during April to June across several years. Because Tompkins county is the best covered area in *eBird*, the learned model is close to the ground truth, according to bird experts at the Cornell Lab of Ornithology. In the second column, we fit the models using the data only from Cortland County in 2015, including that from avicaching locations. We use Cortland County as an example to represent a large number of counties in the United States, where there are few *eBird* submissions. Then in the third column, we further exclude the data collected from Avicaching locations.

As we can see from Fig. 5, the species distribution models in the second column match pretty well in terms of the predicted probabilities with the models in the first column, although they are fitted using much less data. On the contrary, the models in the third column are much worse. Indeed, the log losses improve from 0.44 to 0.30 for Ovenbird, from 0.47 to 0.46 for House Finch, from 0.51 to 0.38 for Wood Thrush and from 0.48 to 0.41 for White-throated sparrow when Avicaching observations are added.

Since the only difference between the models in the second and the third columns is whether the models are learned using the dataset containing observations from *Avicaching* locations, the clear difference in the predictive performance demonstrates the benefit of having data from *Avicaching* locations. From this experiment, we see that *Avicaching* game really helps *eBird* in addressing its ultimate scientific goal.

5   Conclusion

We address the behavior identification problem in two-stage games to reduce the data bias problem in citizen science. We introduce a novel probabilistic behavioral model and show that it is better at capturing noisy human behavior compared to the knapsack model previously used in *Avicaching*, a recently launched gamified application in *eBird*. In addition, the behavioral model can be folded as a set of linear constraints into the bi-level optimization problem for bias reduction, so the whole two-stage game can be solved with a single Mixed Integer Program. We further scale up the encoding to large instances by adding redundant constraints, based on a novel easy-hard-easy phase transition phenomenon. Finally, we also show that the data collected from the *Avicaching* game improves species distribution modeling, therefore it better serves the core scientific goal of citizen science.

References

1. Aggarwal, G., Feder, T., Motwani, R., Zhu, A.: Algorithms for multi-product pricing. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 72–83. Springer, Heidelberg (2004)
2. Anderson, A., Huttenlocher, D.P., Kleinberg, J.M., Leskovec, J.: Steering user behavior with badges. In: 22nd International World Wide Web Conference, WWW (2013)
3. Bacon, D.F., Parkes, D.C., Chen, Y., Rao, M., Kash, I., Sridharan, M.: Predicting your own effort. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 2, pp. 695–702 (2012)
4. Bragg, J., Mausam, Weld, D.S.: Crowdsourcing multi-label classification for taxonomy creation. In: HCOMP (2013)
5. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. **16**, 1190–1208 (1995)
6. Chen, X., Lin, Q., Zhou, D.: Optimistic knowledge gradient policy for optimal budget allocation in crowdsourcing. In: ICML (2013)
7. Chiappone, M.: Coral watch program summary. a report on volunteer and scientific efforts to document the status of reefs in the florida keys national marine sanctuary. The Nature Conservancy, Summerland Key, Florida (1996)
8. Conitzer, V., Garera, N.: Learning algorithms for online principal-agent problems (and selling goods online). In: Proceedings of the 23rd ICML (2006)
9. Conitzer, V., Sandholm, T.: Computing the optimal strategy to commit to. In: Proceedings of the 7th ACM Conference on Electronic Commerce (EC), pp. 82–90 (2006)
10. Endriss, U., Kraus, S., Lang, J., Wooldridge, M.: Incentive engineering for boolean games. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22(3), p. 2602 (2011)
11. Fang, F., Stone, P., Tambe, M.: When security games go green: designing defender strategies to prevent poaching and illegal fishing. In: IJCAI (2015)
12. Gens, R., Domingos, P.M.: Discriminative learning of sum-product networks. In: Advances in Neural Information Processing Systems, pp. 3248–3256 (2012)
13. Gomes, C.P., Selman, B.: Satisfied with physics. Science **297**(5582), 784–785 (2002)
14. Guruswami, V., Hartline, J.D., Karlin, A.R., Kempe, D., Kenyon, C., McSherry, F.: On profit-maximizing envy-free pricing. In: SODA, pp. 1164–1173 (2005)
15. Hartline, J.D., Koltun, V.: Near-optimal pricing in near-linear time. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 422–431. Springer, Heidelberg (2005)
16. Homer, C., Dewitz, J., Fry, J., Coan, M., Hossain, N., Larson, C., Herold, N., Mckerrow, A., Vandriel, J.N., Wickham, J.: Completion of the 2001 national land cover database for the conterminous United States. Photogram. Eng. Remote Sens. **73**(4), 337–341 (2007). http://www.asprs.org/publications/pers/2007journal/april/highlight.pdf
17. Kawajiri, R., Shimosaka, M., Kashima, H.: Steered crowdsensing: Incentive design towards quality-oriented place-centric crowdsensing. In: UbiComp (2014)
18. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning, ICML (2001)
19. Li, H., Tian, F., Chen, W., Qin, T., Ma, Z., Liu, T.: Generalization analysis for game-theoretic machine learning. In: AAAI (2015)

20. Lintott, C.J., Schawinski, K., Slosar, A., et al.: Galaxy zoo: morphologies derived from visual inspection of galaxies from the sloan digital sky survey. Mon. Not. R. Astron. Soc. **389**(3), 1179–1189 (2008). http://dx.doi.org/10.1111/j.1365-2966.2008.13689.x
21. McFadden, D.: Modeling the choice of residential location. In: Spatial Interaction Theory and Residential Location, pp. 75–96 (1978)
22. Paruchuri, P., Pearce, J.P., Marecki, J., Tambe, M., Ordóñez, F., Kraus, S.: Playing games for security: an efficient exact algorithm for solving bayesian stackelberg games. In: AAMAS, pp. 895–902 (2008)
23. Radanovic, G., Faltings, B.: Incentive schemes for participatory sensing. In: AAMAS (2015)
24. Rust, J.: Optimal replacement of gmc bus engines: an empirical model of harold zurcher. Econometrica **55**(5), 999–1033 (1987)
25. Settles, B.: Active learning literature survey. Univ. Wis. Madison **52**(55–66), 11 (2010)
26. Shavell, S.: Risk sharing and incentives in the principal and agent relationship. Bell J. Econ. **10**, 55–73 (1979)
27. Singer, Y., Mittal, M.: Pricing mechanisms for crowdsourcing markets. In: Proceedings of the 22nd International Conference on World Wide Web (WWW) (2013)
28. Singla, A., Santoni, M., Bartók, G., Mukerji, P., Meenen, M., Krause, A.: Incentivizing users for balancing bike sharing systems. In: AAAI (2015)
29. Sullivan, B.L., Aycrigg, J.L., Barry, J.H., et al.: The ebird enterprise: an integrated approach to development and application of citizen science. Bio. Conserv. **169**, 31–40 (2014). http://www.sciencedirect.com/science/article/pii/S0006320713003820
30. Tran-Thanh, L., Huynh, T.D., Rosenfeld, A., Ramchurn, S.D., Jennings, N.R.: Crowdsourcing complex workflows under budget constraints. In: Proceedings of the AAAI Conference, AAAI (2015)
31. Xue, Y., Davies, I., Fink, D., Wood, C., Gomes, C.P.: Avicaching: a two stage game for bias reduction in citizen science. In: Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems, AAMAS (2016)

CP and Biology Track

Constraining Redundancy to Improve Protein Docking

Ludwig Krippahl$^{(\boxtimes)}$ and Pedro Barahona

NOVA-LINCS, DI, FCT-NOVA, 2829-516 Caparica, Portugal
{ludi,pb}@fct.unl.pt

Abstract. Predicting protein-protein complexes (protein docking) is an important factor for understanding the majority of biochemical processes. In general, protein docking algorithms search through a large number of possible relative placements of the interacting partners, filtering out the majority of the candidates in order to produce a manageable set of candidates that can be examined in greater detail. This is a six-dimensional search through three rotational degrees of freedom and three translational degrees of freedom of one partner (the probe) relative to the other (the target). The standard approach is to use a fixed step both for the rotation (typically 10° to 15°) and the translation (typically 1Å). Since proteins are not isotropic, a homogeneous rotational sampling can result in redundancies or excessive displacement of important atoms. A similar problem occurs in the translational sampling, since the small step necessary to find the optimal fit between the two molecules results in structures that differ by so little that they become redundant. In this paper we propose a constraint-based approach that improves the search by eliminating these redundancies and adapting the sampling to the size and shape of the proteins involved. A test on 217 protein complexes from the protein-protein Docking Benchmark Version 5 shows an increase of over 50 % in the average number of non-degenerate acceptable models retained for the most difficult cases. Furthermore, for about 75 % of the complexes in the benchmark, computation time is decreased by half, on average.

Keywords: Protein docking · Geometric search · Constraints

1 Introduction

Protein interactions are crucial in any living organism, since proteins make up most of the biochemical machinery of the cell. Proteins are also the main product of genes and thus lie at the base of the phenotypic expression of the genome and of all metabolism. This is why understanding protein interactions is so important for both theoretical and practical reasons. From the elucidation of biochemical mechanisms to medicine and drug design, predicting how proteins fit together provides useful information. Given the difficulty of studying protein-protein interactions experimentally and the progress in high-throughput methods

of determining individual protein structures [5], modelling protein interactions from the known structures of the interacting partners is bound to remain an important tool for biochemical and medical research.

There are two main approaches to predicting protein-protein complexes, or protein docking. Some algorithms use local search or stochastic sampling methods, such as genetic algorithms [15] or simulated annealing [2], to maximize a scoring function that estimates how favourable the interaction is. This function takes into account several factors such as solvation effects, entropy and electrostatics, which limits the configurations that can be sampled during the search. This is why most docking algorithms use geometric complementarity as a first filtering criterion [1,7,11,12,14], postponing a more detailed evaluation to a second stage, limited to a smaller set of selected candidates. This allows a systematic search through a very large number of possible configurations – on the order of 10^{15} or more – by rotating one of the proteins, the probe, relative to the other, the target, and exploring all translations for each orientation in small steps. Each configuration is then evaluated according to geometric complementarity in order to retain the best candidate models, each being the model for one possible structure for the complex formed by the two interacting proteins. This relatively smaller number of candidate models, with a fixed number that is typically from 10^3 to 10^4 structures, is then scored in more detail and ranked in the evaluation step. This paper focuses on optimizing the search in the filtering step. We use the BiGGER (Bimolecular complex Generation with Global Evaluation and Ranking) docking program, which has a geometric search algorithm based on constraint programming techniques. BiGGER uses constraint propagation to speed up the translational search by eliminating the majority of unproductive configurations, such as those with forbidden overlaps or insufficient surface contact [8] and also to restrict the translational search according to predicted or observed points of contact between the interacting partners [9]. In this paper, we describe the application of the same constraint processing ideas to improve docking results by pruning the rotational search space in a way that accounts for the shape of the protein being rotated and by imposing constraints on the redundancy of the candidates retained. These improvements are integrated in the constraint propagation algorithms that allow BiGGER to search efficiently for good docking model candidates.

1.1   Uniform Rotational Search

The standard approach for the rotational search in exhaustive search docking algorithms is to generate a set of uniformly distributed rotations by rotating the probe molecule around the three orthogonal axis in a constant step, typically ranging from 6° to 15°. Although the implementation details vary, one usual method is to sample the combinations of rotations around the x, y and z axis. If we rotate in steps of 15°, this would mean a total of $12 \times 24 \times 24$ orientations, since there are 24 steps of 15° around each axis but, for one axis, we would only need half a turn to avoid repeating orientations. This results in some degenerate orientations and can be improved by selecting 12×24 uniformly distributed axes

and then rotating 23 steps around each one (the $0°$ and $360°$ rotations around any arbitrary axis all lead to no rotation) and then adding the original orientation, for a total for $12 \times 24 \times 23 + 1$ is 6625 orientations. This is the default uniform rotational search algorithm used in BiGGER. It starts by creating $24 \times 24 = 576$ points uniformly spread on the surface of a sphere of radius 1, using the spiral method [13], and then selecting all those with $z \geq 0$ to define 288 rotation axes.[1] Each rotation axis is then used to generate 23 quaternions specifying the 23 different rotations of $15°$ around the axis, and finally the original orientation (no rotation) is also kept, for the total of 6625 orientations.

1.2   The Search Dilemma

While a finer sampling of both the rotation and translation space is desirable in order to find the best fit between the two proteins, besides the increase in computation time, it also leads to a larger number of very similar configurations. This, in turn, increases the chance that incorrect configurations push all acceptable candidates out of the set of retained models. However, a coarser sample may result in missing the most favourable ways of fitting the proteins and thus not find any acceptable models with a good enough geometric fit to be retained in the filtered set. It is this dilemma that motivates our addition of redundancy constraints. With the method described below, we adapt the rotation sampling to the size and shape of the protein in order that the atom displacement between neighbouring orientations is as close to uniform as possible. This gives us a better control of the trade-off between searching too many or too few orientations. In addition, we constrain the set of retained models to avoid keeping models that are too similar. This way we can do a translation search in small steps of 1Å, in order to better optimize the geometrical fit, but also mitigate the crowding out of acceptable candidates by groups of redundant models.

2   Method

One possibility for optimizing the rotation sampling is to take into account the shape of the probe protein and apply a different constraint to the generation of the rotations. Instead of requiring a constant step, it would be better to require that the maximum displacement of the atoms from one rotation to its nearest neighbour be constant. This would result in a distribution of orientations that is heterogeneous in the angular step but more homogeneous in the atomic distances since proteins are not spherical and can have highly irregular shapes. Another possibility is to prune the rotations searched to reduce the number of orientations used. The rationale for this approach is that, for adequate results, smaller or more globular proteins should need a smaller number of orientations than larger or more irregularly-shaped proteins. The following subsections detail each of these approaches.

---

[1] Or 289 rotation axes, in some previous implementations, depending on exactly how the points are spread with respect to this cutoff for the hemisphere.

2.1   Redundancy in Retained Models

The standard approach to the filtering stage of the docking process is to retain the set of models with the highest scores for the scoring function used in this stage. Given the large search space, this function must be inexpensive to evaluate, and is generally based on estimates of the contact surface [4]. This is the current procedure in BiGGER. However, retaining the highest scoring models can result in redundancies because the step size in the translation search, which in BiGGER is 1Å, can be small enough to result in multiple nearly identical models. On the other hand, a larger translation step could result in missing the most favourable configurations. One way of solving this dilemma is to restrict the models retained so that, of two sufficiently similar models, only the highest scoring one is kept. This consideration, along with the asymmetry of the probe, lead us to the following constraints for reducing redundancy.

2.2   Defining the Constraints

To prevent redundancy in rotation, we define the distance between two rotations as:

Definition 1. *Distance between rotations*
Let S be the set of N points with coordinates $c_1, c_2, ..., c_n$ and ρ_1 and ρ_2 two rotations. The distance between rotations ρ_1 and ρ_2 with respect to S is the largest distance between the pair of images for each point:

$$\mathrm{dist}_S(\rho_1, \rho_2) = \max_{c_i \in S} |\rho_1(c_i) - \rho_2(c_i)|$$

The rotation constraint that reduces the angular redundancy is:

Constraint 1. *Rotation redundancy constraint*
Given a set of points S defining the shape of the probe, a distance parameter δ specifying the smallest non-redundant displacement, and a set R of rotations, then:

$$\forall \rho_i, \rho_{j \neq i} \in R, \mathrm{dist}_S(\rho_i, \rho_j) > \delta$$

Evidently, this constraint is trivial to fulfil with an empty set of rotations. However, we also want to cover all the possible orientations of the probe as well as possible. So the goal is to find the largest set of rotations that respect constraint 1.

To reduce redundancy in retained models, we also add the following constraint:

Constraint 2. *Model redundancy constraint*
Let M be the set of retained models, where each model $m_{t,\rho}$ is determined by the translation vector t and rotation ρ. Given a radius r specifying the redundancy neighbourhood, then:

$$\forall m_{t_i,\rho_i}, m_{t_j,\rho_j} \in M, \rho_i = \rho_j \implies |t_i - t_j| > r$$

This constraint ensures that the final set of candidate models does not retain any pair of models that are mutually redundant, in the sense of having the same orientation and not being farther apart than r. However, by itself, this does not tell us which model to retain. Since we want the best match for the interaction, we need to retain the model with the highest score, which means we have to score all redundant models before filtering. So, unlike the constraint for rotational redundancy, which can greatly prune the search space, the translational redundancy constraint cannot be used to improve computation time.

2.3   Implementing the Constraints

To speed up the calculation, we use a set of 20 atoms to represent the protein shape. These atoms are selected by first picking the atom farthest from the centre and then iteratively picking the atom with the largest distance to the closest atom in those previously picked. This is the set of points over which rotation distances will be calculated. We then create a list of uniformly distributed rotations with a 7.5° step for a total of $24 \times 48 \times 47 + 1 = 54145$ rotations. From this oversampling of the rotational search space we pick the original orientation as the first element of the set of the selected rotations. Then, iteratively, of all rotations for which the minimum distance to any in the selected set is larger than δ, we add the one with the smallest minimum distance to the selected set. We repeat this until no rotation is left that has a minimum distance to the selected set that is larger than δ. This method gives us a distribution of rotations that is more homogeneous in atomic displacement, adapted both to the shape and to the size of the protein, resulting in fewer orientations for smaller proteins and more orientations for larger proteins. For the work related here, the δ value for this rotation constraint was 6Å.

Since this is a greedy optimization, it does not guarantee the maximum possible number of rotations or the ideal distribution. However, it is fast. Building the set of rotations takes from a few seconds to a few minutes for each complex, less than 1 % of the total docking time. Nevertheless, we are working on improving the set of rotations, both by improving the generation algorithm to optimize the distribution of rotations and to fine-tune the δ value.

The model redundancy constraint is implemented by storing a temporary list of the best candidate models during the translational search for a given orientation of the probe. Once the translational search for that orientation is complete, the sorted candidates are examined starting from the model with the largest surface contact and, whenever the program adds one model to the final list of selected models, it removes all models in the temporary list for which the translation vector is within 2.5Å of the inserted model in all three coordinates. This parameter also needs to be optimized by experimenting with different values, as well as the decision to use a cubic neighbourhood instead of a spherical one. These adjustments are still work in progress.

Figure 1 outlines the BiGGER docking algorithm. The rotational redundancy constraints are applied in line 2, adjusting the sample of orientations to the shape of the probe protein, which can significantly reduce the search space in

most cases without sacrificing the quality of the results. Then, for each orientation, the probe is rotated appropriately and the main BiGGER constrained search function is run for the translations (line 7). This translational search uses information on the shape of the proteins and the minimum score of the models retained so far to prune the configurations to analyse [8]. The resulting candidate models are kept in a temporary set to which the model redundancy constraint is applied (line 8) by keeping only the highest scoring model $m_{t,\rho}$ out of any set of models that are redundant with respect to $m_{t,\rho}$, according to Definition 2. This constraint actually reduces the pruning of the translational search space, indirectly, because BiGGER prunes this search using the score of the lowest scoring model and discarding redundant models will keep this value lower than it would be otherwise. However, the cost of this constraint is small (around 5 % of the total computation time) and is compensated by the improvement in the results and by the pruning effects of the rotational constraint, in most cases, as the benchmark tests show.

```
 1: function DOCK(target, probe)
 2:     rotations ← constrain_rotation(probe)
 3:     candidates ← [ ]
 4:     for rotation in rotations do
 5:         oriented_probe ← rotate(probe, rotation)
 6:         min_score ← lowest(candidates)
 7:         temp_set ← constrained_translation(target, oriented_probe, min_score)
 8:         temp_set ← remove_redundant(temp_set)
 9:         candidates ← merge(temp_set, candidates)
10:     end for
11:     return candidates
12: end function
```

Fig. 1. Outline of how the redundancy constraints proposed in this paper (bold) fit into the BiGGER constraint-based docking algorithm.

2.4   Benchmark Tests

To test our method, we used the protein-protein Docking Benchmark Version 5 [16]. This is a benchmark of 230 cases of unbound docking of protein complexes. However, these 230 examples only span 225 protein complexes, as five of the 230 examples are additional binary complexes drawn from some large complexes in the pool of 225. Of these 225 complexes, in 217 the length of the probe was distributed with an approximately normal distribution with an average of 55Å and a standard deviation of 13Å, and ranging from 25Å through 86Å. The length of the probe was measured as the largest distance between any pair of atoms in the protein. This is an important measure because it affects both the number of orientations necessary to adequately sample the rotational space and the size

of the translational search space. In the other eight complexes the probe length was spread out from 98Å to 141Å. This sparse set of a few but extreme outliers caused difficulties in aggregating the results as a function of the length of the probe, which was necessary for judging the effects of the constraints in different conditions. For this reason, we ended up rejecting these eight complexes, leaving us with a benchmark set of 217 different complexes, each involving two proteins.

The protein-protein Docking Benchmark provides the unbound structures and the target complexes recreated by rigidly fitting the unbound partners to the known complex structure. These were the target complexes we used to evaluate the performance of the docking algorithm. In some cases, there are significant conformational changes between the bound and unbound proteins. Using these target complexes provided by the benchmark, we can evaluate the rmsd[2] of the docking predictions without adding the irreducible remaining error due to the conformational changes the proteins undergo when interacting, which can be as high as 8Å. Nevertheless, these conformational changes still add to the difficulty of the unbound docking, since we are trying to predict the correct fit of proteins that are not in the ideal conformation for fitting together, as would generally be the case in a real application. In addition to using the unbound conformations, we also rotated each probe protein randomly before docking. Other than the constraints described here, the docking predictions were run with the default parameters used by BiGGER and retaining a set of 5000 models.

To evaluate the results, we considered a model to be acceptable if the rmsd value computed for the probe was below 10Å and the rmsd value for the atoms at the interface was below 4Å. The interface is the set of atoms within 5Å of any atom of the other partner. These are criteria used in the CAPRI programme for assessing predictions of protein interactions [6].

3   Results and Discussion

Figure 2 shows the relative effect of using both redundancy constraints compared to the base search algorithm of uniform rotation sampling and not discarding redundant models. These relative values are computed dividing the values for the docking runs with the redundancy constraints by the respective values for the base docking runs. The lines are smoothed using a Gaussian kernel with 5Å of standard deviation for each data point. The failure rate is the proportion of complexes for which no acceptable model was retained in the final set of 5000 candidates. The relative failure rate is always below one, meaning that the redundancy constraints result in a lower failure rate than the corresponding docking runs using the base algorithm. The relative number of non-redundant acceptable models is nearly always higher than one, which shows that the redundancy constraints increase the average number of acceptable models retained in the final set. The relative time is below one for the first three quartiles, meaning that, in 75 % of the cases, the number of orientations with the fixed displacement of 6Åis

---

[2] The square root of the mean of the squared atomic deviations, in Ångstrom.

smaller than the default number of orientations. However, since they are better distributed, even with a smaller number of orientations to search the results are better than the results of the base method. For larger probe sizes, the number of rotations is larger than 6625, the default used by BiGGER. In this quartile, computation times can increase significantly. However, the average number of acceptable models retained also increases significantly, suggesting that the base set of 6625 orientations was inadequate for larger probes.
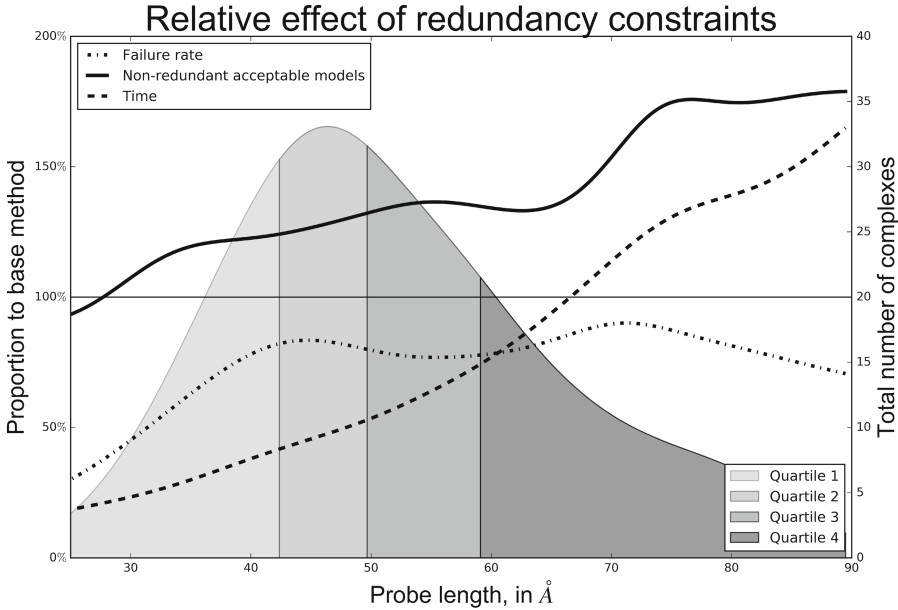

Fig. 2. The lines show the relative effects of the redundancy constraints on failure rate, average number of non-redundant acceptable models retained and computation time as a function of probe length. The values are in proportion to the values obtained with the base algorithm. The shaded curve shows the length distribution of the probes in the benchmark examples, divided into quartiles. All plots were smoothed using a Gaussian kernel with $\sigma = 5\mathring{A}$.

Table 1 shows the values comparing, for each quartile, four different cases. The *Base* case is the basic BiGGER algorithm without any redundancy constraints. The *Models* case uses Constraint 2 to prevent redundancy on the models retained. *Rotations* uses Constraint 1 to adapt the set of rotations to the shape and size of the probe, so that no two different orientations result in a maximum atomic displacement below 6Å. Finally, *Full* uses both constraints. The results are aggregated by each quartile of the distribution of probe lengths, with the last column showing the averages for all 217 test complexes. Each value is the average value plus or minus the standard deviation for the average estimated by bootstrapping [3] with 10,000 replicas.

Table 1. Effect of redundancy constraints, by quartile

Quartile	Average number of non-redundant models by complex				
Quartile	0-25 %	25-50 %	50-75 %	75-100 %	All
Base	13.0 ± 2.7	4.7 ± 0.7	3.8 ± 0.7	2.6 ± 0.4	6.1 ± 0.8
Model	18.9 ± 3.8	6.8 ± 1.2	5.1 ± 0.9	3.4 ± 0.5	8.6 ± 1.1
Rotation	9.3 ± 1.2	4.1 ± 0.5	3.6 ± 0.5	3.1 ± 0.6	5.1 ± 0.4
Full	14.6 ± 1.8	5.9 ± 0.8	5.3 ± 0.7	4.1 ± 0.7	7.5 ± 0.6
Quartile	Percentage rate of failed predictions				
Quartile	0-25 %	25-50 %	50-75 %	75-100 %	All
Base	$9\% \pm 4\%$	$13\% \pm 5\%$	$28\% \pm 6\%$	$33\% \pm 6\%$	$21\% \pm 3\%$
Model	$9\% \pm 4\%$	$7\% \pm 4\%$	$15\% \pm 5\%$	$24\% \pm 6\%$	$14\% \pm 2\%$
Rotation	$9\% \pm 4\%$	$11\% \pm 4\%$	$22\% \pm 6\%$	$37\% \pm 7\%$	$20\% \pm 3\%$
Full	$5\% \pm 3\%$	$13\% \pm 5\%$	$20\% \pm 5\%$	$28\% \pm 6\%$	$17\% \pm 3\%$
Quartile	Average computation time by complex (hours)				
Quartile	0-25 %	25-50 %	50-75 %	75-100 %	All
Base	0.7 ± 0.0	1.2 ± 0.1	1.4 ± 0.1	2.7 ± 0.2	1.5 ± 0.1
Model	0.7 ± 0.0	1.2 ± 0.1	1.5 ± 0.1	2.8 ± 0.2	1.5 ± 0.1
Rotation	0.2 ± 0.0	0.5 ± 0.0	0.8 ± 0.1	2.9 ± 0.3	1.1 ± 0.1
Full	0.2 ± 0.0	0.6 ± 0.0	0.9 ± 0.1	3.2 ± 0.4	1.2 ± 0.1

This table compares the basic BiGGER algorithm with docking impos-
ing the constraint on model redundancy, the constraint on rotation
redundancy and both. The results are aggregated by quartile of the
distribution of probe lengths. The last column shows the aggregate
values for all 217 test complexes.

The number of non-redundant complexes retained in the final set of 5000 can-
didates is relevant for estimating the difficulty of identifying the correct models
within this set. Other factors being equal, after the second stage of evaluat-
ing this set with a more detailed scoring function, the greater the number of
acceptable models present the easier it should be to pinpoint the correct com-
plex structure. Looking at the four quartiles in the distribution of probe lengths,
we can see that docking small probes is easier, resulting in around 10 to 20
non-redundant acceptable models. The failure rate, which is the percentage of
complexes for which no acceptable model was retained in this final set of 5000, is
also lowest for smaller probes. Thus, in this quartile, the most significant gain by
combining the two constraints is in computation time, which is reduced to nearly
a quarter of the time for the unconstrained docking. Using the rotation redun-
dancy constraint alone can give us this performance improvement but results in
a lower average number of acceptable models, although the failure rate is not
significantly different.

On the second quantile, the results are similar, differing only in that the time
decrease is less marked, at around 50 %, and the increase in the average number

of acceptable models seems to be greater, though still not significant. On the third quartile, however, the average number of non-redundant acceptable models retained is significantly higher for the dockings with both constraints than it is with the unconstrained dockings, and slightly higher than any of the constraints alone. The best explanation for this seems to be the antagonistic effects of increasing the sampling of the rotation space. While, on the one hand, such an increase increases the probability of not missing acceptable models during the search, on the other hand, increasing the sampling density also increases the number of incorrect models which can displace acceptable candidates from the final set of 5000 models to retain. This seems to be why the two constraints combined outperform either one alone, with the model redundancy constraint mitigating the negative effects of increasing the rotational sampling as the probe length increases, while the rotational redundancy constraint leads to a sample of orientations better suited to the shape and size of the probe.

On the last quartile this improvement in the average number of acceptable models retained is even more marked, being over 50 %. This is particularly relevant because these are the hardest complexes to predict. The tendency for larger proteins to suffer greater conformational changes and the larger number of models to filter during the search result in significantly fewer acceptable models being retained in the final set and increases the chances that none will be retained. This is clear from the absolute values in the table. In these conditions, the redundancy constraints provide an important advantage in the slight decrease in failure rate and, in particular, in the significant increase in the number of acceptable models retained. In these more difficult complexes, however, the number of orientations sampled using the rotational redundancy constraint becomes larger than the default of 6625, and thus the computation time also increases.

4   Conclusions and Future Work

This paper presents an improvement on the search and filtering stage of protein docking using principles of constraint programming. By imposing constraints that prevent redundancies in the rotational search and the retention of candidate models, the average number of acceptable models increases, failure rates decrease slightly and computation times decrease in 75 % of the cases due to the pruning of the rotational search space. Of greater advantage, the average number of acceptable models retained increases significantly in the quartile corresponding to the most difficult complexes to model, where an improvement in the quality of the results is most important. Furthermore, since this is an algorithmic improvement that requires no additional data, it can be combined with other constraints that BiGGER can use, such as symmetry constraints [10] or constraints derived from predicted contacts [9].

There are still some open issues that we are currently exploring. The values of 6Å for the displacement in the rotation constraint and 2.5Å for the redundancy of the models retained seem intuitively reasonable but must be systematically compared to alternative values in order to optimize these constraints. Furthermore, it is quite possible that the optimal values depend on the size of the probe,

especially for larger probes. We expect that the results presented here can be improved by fine-tuning these parameters and, possibly, adapting them to the size of the proteins involved. This work also focused on the effects of using these constraints on the search and filtering stage. We are currently working on testing these modifications on the full BiGGER docking pipeline currently under development, which begins with the prediction of likely contacts from sequence data, proceeds with the constrained dockings and ends with the screening of the retained complexes using more detailed scoring function.

The source code for the implementation of the methods described here is available as part of the Open Chemera Library, at https://github.com/lkrippahl/Open-Chemera. The source code is published in the public domain and is free of any copyright restrictions.

References

1. Chen, R., Li, L., Weng, Z.: ZDOCK: an initial-stage protein-docking algorithm. Proteins Struct. Funct. Bioinf. **52**(1), 80–87 (2003)
2. Dominguez, C., Boelens, R., Bonvin, A.M.J.J.: HADDOCK: a protein-protein docking approach based on biochemical or biophysical information. J. Am. Chem. Soc. **125**(7), 1731–1737 (2003)
3. Efron, B.: Bootstrap methods: another look at the jackknife. Ann. Stat. **7**(1), 1–26 (1979)
4. Halperin, I., Ma, B., Wolfson, H., Nussinov, R.: Principles of docking: an overview of search algorithms and a guide to scoring functions. Proteins Struct. Funct. Bioinf. **47**(4), 409–443 (2002)
5. Hura, G.L., Menon, A.L., Hammel, M., Rambo, R.P., Ii, F.L.P., Tsutakawa, S.E., Jenney Jr., F.E., Classen, S., Frankel, K.A., Hopkins, R.C., et al.: Robust, high-throughput solution structural analyses by small angle x-ray scattering (SAXS). Nat. Methods **6**(8), 606–612 (2009)
6. Janin, J.: Assessing predictions of protein-protein interaction: the CAPRI experiment. Protein Sci. **14**(2), 278–283 (2005)
7. Katchalski-Katzir, E., Shariv, I., Eisenstein, M., Friesem, A.A., Aflalo, C., Vakser, I.A.: Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. Proc. Nat. Acad. Sci. **89**(6), 2195–2199 (1992)
8. Krippahl, L., Barahona, P.: Applying constraint programming to rigid body protein docking. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 373–387. Springer, Heidelberg (2005)
9. Krippahl, L., Barahona, P.: Protein docking with predicted constraints. Algorithms Mol. Biol. **10**(1), 9 (2015)
10. Krippahl, L., Barahona, P.: Symmetry constraints for modelling homo-oligomers. In: 11th Workshop on Constraint Based Methods for Bioinformatics (2015)
11. Palma, P.N., Krippahl, L., Wampler, J.E., Moura, J.J.: Bigger: a new (soft) docking algorithm for predicting protein interactions. Proteins **39**(4), 372–384 (2000)

12. Roberts, V.A., Thompson, E.E., Pique, M.E., Perez, M.S., Ten Eyck, L.: Dot2: macromolecular docking with improved biophysical models. J. Comput. Chem. **34**(20), 1743–1758 (2013)
13. Saff, E.B., Kuijlaars, A.B.: Distributing many points on a sphere. Math. Intell. **19**(1), 5–11 (1997)
14. Schneidman-Duhovny, D., Inbar, Y., Polak, V., Shatsky, M., Halperin, I., Benyamini, H., Barzilai, A., Dror, O., Haspel, N., Nussinov, R., et al.: Taking geometry to its edge: fast unbound rigid (and hinge-bent) docking. Proteins Struct. Funct. Bioinf. **52**(1), 107–112 (2003)
15. Taylor, J.S., Burnett, R.M.: Darwin: a program for docking flexible molecules. Proteins: Struct. Funct. Bioinf. **41**(2), 173–191 (2000)
16. Vreven, T., Moal, I.H., Vangone, A., Pierce, B.G., Kastritis, P.L., Torchala, M., Chaleil, R., Jiménez-García, B., Bates, P.A., Fernandez-Recio, J., et al.: Updates to the integrated protein-protein interaction benchmarks: Docking benchmark version 5 and affinity benchmark version 2. J. Mol. Biol. **427**(19), 3031–3041 (2015)

Guaranteed Weighted Counting for Affinity Computation: Beyond Determinism and Structure

Clément Viricel[1,2], David Simoncini[1], Sophie Barbe[2], and Thomas Schiex[1(✉)]

[1] MIAT, Université de Toulouse, INRA UR 875, Castanet-Tolosan, France
{clement.viricel,david.simoncini,thomas.schiex}@toulouse.inra.fr
[2] LISBP, Université de Toulouse, CNRS, INRA, INSA, Toulouse, France
sophie.barbe@insa-toulouse.fr

Abstract. Computing the constant Z that normalizes an arbitrary distribution into a probability distribution is a difficult problem that has applications in statistics, biophysics and probabilistic reasoning. In biophysics, it is a prerequisite for the computation of the binding affinity between two molecules, a central question for protein design. In the case of a discrete stochastic Graphical Model, the problem of computing Z is equivalent to weighted model counting in SAT or CSP, known to be #P-complete [38]. SAT solvers have been used to accelerate guaranteed normalizing constant computation, leading to exact tools such as `cachet` [33], `ace` [8] or `minic2d` [28]. They exploit determinism in the stochastic model to prune during counting and the dependency structure of the model (partially captured by tree-width) to cache intermediary counts, trading time for space. When determinism or structure are not sufficient, we consider the idea of discarding sufficiently negligible contributions to Z to speedup counting. We test and compare this approach with other solvers providing deterministic guarantees on various benchmarks, including protein binding affinity computations, and show that it can provide important speedups.

1 Introduction

Graphical models [12] are sparse representations of highly dimensional multivariate distributions that rely on a factorization of the distribution in small factors. When variables are discrete, graphical models cover a variety of mathematical models that represent joint discrete distributions (or functions) that can be either Boolean functions (*e.g.,* in propositional satisfiability SAT and constraint satisfaction CSP), cost functions (as in partial weighted MaxSAT and Cost Function Networks [9]) or probability distributions (in stochastic models such as Markov Random Fields and Bayesian networks).

Typical queries on such graphical models are either optimization or counting queries (or a mixture of these). In optimization queries, we look for an assignment that maximizes the joint function, *i.e.,* a model in SAT, a solution in CSP or a

Maximum a posteriori assignment (MAP) in a Markov Random Field (MRF). All these problems have an associated NP-complete decision problem.

Counting problems are central in stochastic graphical models because they capture the computation of marginal probabilities on subsets of variables and the computation of the normalizing constant Z that is required to define a probability distribution from the non-normalized distribution of Markov Random Fields. This difficult problem requires a summation over an exponential number of elementary terms and is #P-complete [38]. As shown by [37], one call to a #-P oracle suffices to solve any problem in the Meyer-Stockmeyer polynomial hierarchy in deterministic polynomial time, an indication that it could be outside of the PH.

Computing Z is a central problem in statistics (*e.g.,* for parameter estimation in MRFs), for Bayesian network processing (to account for evidence) and is also crucial in statistical physics where it is called the partition function. A typical domain where partition function computation can be extremely useful is computational protein design. Indeed, the affinity of a protein for a specific target molecule can be estimated by modeling both molecules as MRFs representing physics force fields and by computing the two partition functions: one for the bound protein and target and another for the same molecules in unbound state [35].

For these reasons, various approaches have been designed to tackle this problem. The Mean-Field algorithm [17], Tree-reweighted Belief Propagation [40] as well as more recent proposals [25] have been proposed, but they do not offer any formal guarantee on the quality of the approximation they produce, except in very special cases. Monte-Carlo methods including Markov Chain Monte Carlo methods [16] offer asymptotic convergence /s, but convergence is impractically slow. Indeed, there are recent significant examples showing that the time needed for Monte Carlo methods to converge can be easily under-estimated [36]. Practical MCMC based tools also rely on heuristics that destroy these theoretical guarantees. More recent stochastic methods exploiting universal hashing functions offer "Probably Approximately Correct" (PAC) estimators [7,14]. Here, a bound δ on the probability that the estimation does not lie within a $(1+\varepsilon)$ ratio of the true value is set and a corresponding estimation produced.

Finally, different methods, mostly based on SAT-solvers, have been defined that can perform exact weighted model counting (#SAT) with deterministic guarantees, a problem to which the problem of computing Z can be easily reduced. To avoid the exponential blowup in the number of terms to add, solvers providing deterministic guarantees rely on two independent ideas: exploiting determinism (zero weights) to prune regions of the space that do not contribute to the sum, and exploiting independence which may be detected at the graphical model structure level, as captured by its tree-width, but also at a finer level as context-sensitive independence [33][1]. Independence enables caching of intermediate counts that can

---

[1] They may also exploit the fact that counting the number of models of a valid formula is easy. This requires to check for validity, something that modern CDCL solvers do not do anymore.

be factored out and lead to exponential time savings at the cost of memory. The very same ideas are also exploited in knowledge compilers that may compile graphical models or SAT formulas to languages on which counting becomes easy [8,28].

In this paper, we explore the possibility of preserving the *deterministic* guarantees of exact solvers and explore a new source of pruning that may be present even when determinism or independence are too limited to allow for exact counting: detecting and pruning regions for which it is possible to prove, at limited cost, that they contain an amount of weight which is too small to significantly change the computed value of Z. Instead of providing a PAC guarantee, our algorithm provides an approximation of the normalizing constant that is guaranteed to lie within a ratio of $1 + \varepsilon$ of the true value (with probability 1), a guarantee that none of the PAC randomized algorithms above can provide in finite time.

Our initial motivation for computing Z lies in Computational Protein Design (CPD). The aim of CPD is to design new proteins that have desirable properties which are not available in the existing catalog of known proteins. One of these properties is the *affinity* between a protein and another molecule (such as another protein, a peptide, an amino-acid, a small organic molecule, etc...). The binding affinity gives an indication of the likelihood that two molecules will prefer to bind together rather than remain dissociated and thus that a protein will be likely to bind to another molecule of interest. Proteins can be described as a set of bound atoms subjected to a number of atom scale forces captured by a pairwise force field defining a Markov Random Field [29]. From this MRF, the binding affinity can be estimated by computing the ratio of the partition functions of the molecules in bound and unbound states [15,35].

In the rest of the paper, after introducing our notations and the binding affinity computation problem, we present the Z_ε^* algorithm, a variant of Branch and Bound targeted at counting instead of optimizing. Z_ε^* relies on the availability of a local upper bound on Z. We then consider different simple, fast, safe and incremental upper bounds on Z, integrate them in Z_ε^* and compare them to exact counting tools on two categories of benchmarks: general benchmarks extracted from the UAI and Probabilistic Inference (PIC'2011) challenges and partition function computation problems appearing as sub-problems of binding affinity computation on real proteins. Surprisingly, despite a very limited caching strategy, the resulting algorithm is able to outperform exact solvers on a variety of problems and is especially efficient on CPD-derived problems. Because Z_ε^* relies on a new source of pruning, its underlying principle and associated bounds can be immediately used to improve existing SAT-based counters using Max-SAT bounds, which are closely related to local consistencies in Cost Function Networks [4,23,24].

2   Background

A Markov Random Field defines a joint probability distribution over a set of variables as a factorized product of local functions, usually denoted as potential functions.

Definition 1. *A discrete Markov Random Field (MRF) is a pair (X, Φ) where $X = \{1, \ldots, n\}$ is a set of n random variables, and Φ is a set of potential functions. Each variable $i \in X$ has a finite domain D^i of values that can be assigned to it. A potential function $\phi_S \in \Phi$, with scope $S \subseteq X$, is a function $\phi_S : D^S \mapsto \mathbb{R} \cup \{\infty\}$ where D^S denotes the Cartesian product of all D^i for $i \in S$.*

The energy or potential of an assignment $t \in D^X$ is denoted as $E(t) = \sum_{\phi_S \in \Phi} \phi_S(t[S])$ where $t[S]$ is the projection (or restriction) of t to the variables in S. Notice that this definition shows that an MRF is essentially equivalent to a Cost Function Network (or WCSP [9]). A tuple $t \in D^S$ will be represented as a set of pairs $\{(i, t[i]) \mid i \in S\}$.

The probability of a tuple $t \in D^X$ is then defined as:

$$P(t) = \frac{\exp(-E(t))}{\sum_{t' \in D^X} \exp(-E(t'))}$$

The normalizing constant below the fraction is usually denoted as Z. The potential ϕ_S are called energies, in relation with statistical physics. An assignment with minimum energy has therefore maximum probability. With pairwise potentials ($|S| \leq 2$), an MRF defines a graph with variables as vertices and potential scopes S as edges. In the rest of this paper, for the mere sake of simplicity and w.l.o.g., we assume pairwise MRFs including also unary potential functions and a constant $\phi_\varnothing$ potential function. We denote by d the maximum domain size and e the number of pairwise potential functions. Using table representations, a pairwise MRF requires $O(ed^2)$ space to be represented.

Note that Bayesian networks can be seen as specific MRFs enforcing a local normalization condition of potentials and a specific DAG-base graph structure, that together guarantee that $Z = 1$. As soon as evidence (observations) change the domain of the variables however, Bayesian networks become unnormalized and computing Z becomes #P-complete in general.

2.1   Computational Protein Design and Binding Affinity

Proteins are linear chains of small molecules called "amino-acids". There are 20 natural different amino-acids. All amino-acids share a common core and the cores of all successive amino-acids in a proteins are linked together to form a linear chain, called the protein backbone. Each amino-acid also has a variable side-chain which chemical nature defined the precise amino-acid used. This lateral chain is highly flexible. The structure of a protein in 3D-space is therefore characterized by the shape of the linear chain itself (the backbone), and the specific spatial orientation of all side-chains, at each position of the chain. Proteins are universally present in the cells of all living organisms and perform a vast array of functions including catalyze, signaling, recognition, transporting, repair... Proteins differ from one another primarily in their sequence of amino-acids which usually results in protein folding into a specific 3D structure that determines its function. The characteristic of proteins that also allows their diverse set of functions is their ability to bind other molecules, with high affinity and specificity. See [1,5] for an intrduction to proteins targeted at the CP audience.

Proteins have a relatively stable general shape. The relative stability of a molecule in a given conformation can be evaluated by computing its energy, lower energy states being more stable. This energy is derived from various molecular forces including bond angles, electrostatic forces, molecular clashes and distances. It can be computed using existing force fields such as Amber [29], the one used in our experiments. Notice that molecular clashes – interpenetrating atoms – may generate infinite energies *i.e.,* determinism.

Despite a plethora of functionalities of proteins, there is still an ever-increasing demand for proteins endowed with specific properties of interest for many applications (in biotechnology, synthetic biology, green chemistry and nanotechnology) which either do not exist in nature or have yet not been found in the biodiversity. To this end, Computational structure-based Protein Design (CPD) has become a key technology. By combining physico-chemical models governing relations between protein amino-acid composition and protein 3D structure with advanced computational algorithms, CPD seeks to identify one or a set of amino-acid sequences that fold into a given 3D structure and possess the targeted properties. This *in silico* search for the best sequence candidates opens up new possibilities to better guide protein engineering by focusing experimentation on the relevant sequence space for the desired protein function and thereby reducing the size of mutant libraries that need to be built and screened. In recent years, CPD has experienced important success, especially in the design of therapeutic proteins [27], novel enzymes [31], protein-protein interfaces [18,32], and large oligomeric ensembles [19]. Nevertheless, the computational design of proteins with defined affinity for a given molecule (such as a small organic, a peptide, another protein...) which is essential for large range of applications, continues to present challenges.

A traditional approach to model proteins in CPD is to assume that their backbone is totally rigid and that only side-chains move, each side-chain being able to adopt a discrete set of most likely conformations defined in a so-called "rotamer" library (see Fig. 1). We use the Penultimate rotamer library [26].
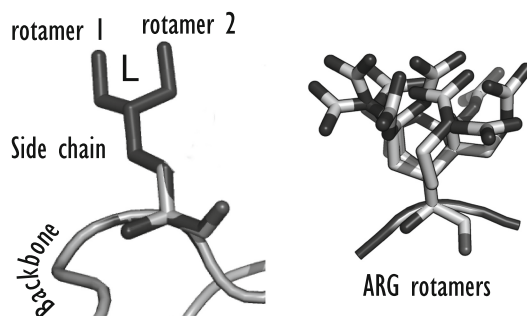

Fig. 1. A local view of a protein with a backbone and two acid side-chain reorientations (rotamers) for a given amino-acid (L = Leucine). A typical rotamer library for another amino-acid is shown on the right (ARG = Arginine).

With one variable per side-chain, each with a domain equal to the set of available rotamers for this side-chain and a pairwise decomposable energy function such as Amber force field, a protein naturally defines a pairwise MRF with a rather dense graph. The partition function Z of this MRF captures important properties of the protein. Specifically, the association constant (or binding constant) is used to describe the affinity between a protein and a ligand (a protein or another molecule of interest). This association constant can be estimated by computing the partition function of the two molecules in bound and unbound states. The ratio of these two partition functions being proportional to their affinity.

From a computational point of view, an important property of proteins of interest is that their general shape is stable which means that the proportion of low energy (or high probability) states among the exponential number of possible states is likely to be very small. On the opposite side of the energy scale, the infinite energies created by molecular clashes means that there will be states with 0 probability. This is favorable for exact solvers that can exploit determinism to speedup Z computation. It however means that CPD instances will exhibit unbounded *tilt* (defined in [7] as the ratio $\tau = \frac{\max_{t \in D^X} P(t)}{\min_{t \in D^X} P(t)}$). This situation is not ideal for the WeightMC PAC algorithm which requires a finite upper-bound on τ to run in finite time.

3   Guaranteed Counting

Because it is rarely (if ever) needed to compute a probability or a partition function with an absolute precision (which is also inherently limited by finite representations), we consider the general problem of computing an ε- approximation $\hat{Z}$ of Z, *i.e.,* such that:

$$\frac{Z}{1+\varepsilon} \leq \hat{Z} \leq Z \tag{1}$$

Such approximation allows us to compute an estimate $\hat{P}(t) = \frac{\exp(-E(t))}{\hat{Z}}$ such that $P(t) \leq \hat{P}(t) \leq (1+\varepsilon)P(t)$. In the context of #-SAT, it has been shown that providing such relative approximations remains intractable for most of the known SAT polynomial classes [30]. As we will see, it can however be exploited to prune during polynomial space depth-first tree-search based counting and sometimes provide important speedups.

Assuming that for any MRF, and any assignment t of *some* of its variables, we can compute an upper bound $Ub(t)$ of the partition function of the MRF where variables are assigned as in t, the Depth First Branch and Bound schema used for exactly solving optimization problems on cost function networks [1,9] can be adapted to compute Z [39].

The algorithm simply explores the tree of all possible assignments of the MRF, starting with the whole set of unassigned variables (in V), choosing an unassigned variable (line 1), trying all possible values. When all variables are

Function $Z_\varepsilon^*(t, V)$

1   **if** $V = \varnothing$ **then**

2 $\hat{Z} \leftarrow \hat{Z} + \exp(-E(t))$;

   else

3   Choose $i \in V$;

     for $a \in D^i$ **do**

       $t' \leftarrow t \cup \{(i, a)\}$;

4   **if** $(U + Ub(t') + \hat{Z} \leq (1 + \varepsilon)\hat{Z})$ **then**

5 $U \leftarrow U + Ub(t')$;

       else

         $Z_\varepsilon^*(t', V - \{i\})$;

Algorithm 1. Guaranteed approximate counting. Initial call: $Z_\varepsilon^*(\varnothing, X)$. U and $\hat{Z}$ are global variables initialized to 0.

assigned (line 1), the contribution of the complete assignment t is accumulated in a running count which will eventually define the approximation $\hat{Z}$ (line 2). However, branches which provide a sufficiently small mass of probability (as estimated by $Ub(t')$) are pruned and this overestimation of the neglected mass is accumulated in U (line 5). Because pruning may occur, eventually, $\hat{Z}$ will be a lower bound of Z.

Theorem 1. Z_ε^* *terminates and returns an ε-approximation of Z.*

Proof. The termination follows from the fact that Z_ε^* explores a finite tree. We now show that the algorithm always provides a ε-approximation. When the algorithm finishes, all the assignments have either been explored (line 2) and counted or pruned (line 5). Since U is the sum of all the upper bounds on the mass of probability in all pruned branches, we have that $\hat{Z} + U \geq Z$. Initially, $\hat{Z} = U = 0$ and the invariant $\hat{Z} \geq \frac{\hat{Z} + U}{1 + \varepsilon}$ holds. The test at line 4 guarantees that this invariant still holds at the end of the algorithm. Therefore $\hat{Z} \geq \frac{Z}{(1 + \varepsilon)}$. □

While inspired by Depth First Branch and Bound (DFBB) that provides polynomial space complexity, this algorithm behaves differently from it. In DFBB, for a fixed order of exploration, when the local bound used for pruning (here $Ub(t)$) is tighter, less nodes are explored. This property is lost in Z_ε^*. Indeed, it is easy to imagine a scenario where a tight bound $Ub(t)$ will lead to more nodes being explored than using a weaker $Ub'(t)$: imagine that search has started and collected a mass $\hat{Z} = 1$ and $U = 0$ for either bounds. Then comes a subtree of small size for which $Ub(t) = \varepsilon$ while $Ub'(t) \gg \varepsilon$. This subtree will be pruned by $Ub(t)$ leading to $U = \varepsilon$ but instead will be enumerated with $Ub'(t)$ preserving $U = 0$. In this context, the algorithm using the tight $Ub(t)$ is not allowed to prune anymore in the immediate future: if the forthcoming leaves all have very small probability mass, it will be forced to visit all of them while

the algorithm using $Ub'(t)$ preserved some margin and may be able to skip a significant fraction of them.

Indeed, similarly to what happens with the α-β algorithm [20], the order in which leaves are explored may have a major effect on the algorithm efficiency. Let us assume that we have a perfect $Ub(t)$ and that the leaves of the tree have exponentially decreasing mass of probability, the ith visited leaf having a mass of $\varepsilon^{i-1}, \varepsilon < 1$ (such an extreme distribution of probability mass may seem unlikely, but corresponds to linearly increasing energies). In this case, the first leaf bears more mass than all the rest of the tree and the Z_ε^* algorithm would visit just one leaf. If the inverse ordering of leaves is assumed, the algorithm will have to explore all leaves. It seems therefore important to collect highest masses first. The polynomial space complexity of DFBB comes however with strict constraints on the order of exploration of leaves and best-first algorithms that could overcome this restriction would lead to worst-case exponential space complexity. Interesting future work would be to use the recent highly flexible any-space Branch and Bound algorithm HBFS [2] to improve the leaf ordering within bounded space.

However, contrary to what happens with optimization, even an exact upper bound and a perfect ordering does not guarantee that only one leaf needs to be explored. If we instead assume a totally flat energy landscape, with all leaves having the same energy, Z_ε^* will have to explore a $\frac{1}{1+\varepsilon}$ fraction of the leaves just to accumulate enough mass in $\hat{Z}$ to prune.

Overall, it is important to realize that Z_ε^* needs to achieve two goals:

1. collect probability masses on a potentially very large number of complete assignments to compute a suitable approximation
2. exploit its upper bound to prune the largest possible part of the tree

The first goal could be achieved by existing algorithms producing an exhaustive list of the m-best assignments [13] or all assignment within a threshold of the optimum (a service that any DFBB-based optimization system provides for free). These algorithms use bounds on the *maximum* probability instead of the total probability mass which leads to stronger pruning and potentially higher efficiency than Z_ε^* but do not provide any guarantee since the number m of assignments that would need to be enumerated to provide ε-approximation is unknown.

Because a potentially very large number of probability masses need to be collected, a very fast search is required. To accelerate it, we equip Z_ε^* with a very simple form of "on the fly" caching: at any node during the search, we eliminate any variable which is either assigned or of bounded degree as proposed initially for optimization [22], but using sum-product variable elimination [11]. This caches all the influence of the eliminated variable in a temporary (trailed) potential function. This means that the leaves of the search tree will be subproblems with bounded tree-width that may represent an exponential number of assignments. This naturally makes Z_ε^* related to the `vec` weighted counting algorithm, an anytime MRF counter based on w-cutsets (vertex cutset which if assigned leave a w-tree) and variable elimination over w-trees [11].

The second goal is to prune the largest possible part of the tree search. However, since the first goal requires a very fast search algorithm, using a powerful but computationally expensive bound is probably doomed to fail. For this reason, we have considered simple fast incrementally updated upper bounds by borrowing recent optimization bounds [9] which are known to work well in conjunction with Depth First Search.

3.1   Bounds for Guaranteed Counting

For any MRF, we define a first upper bound on Z denoted by Ub_1.

$$Z \leq Ub_1 = \left(\prod_{\phi_S, |S|<2} \sum_{t \in D^S} \exp\left(-\phi_S(t)\right) \right) \cdot \left(\prod_{\phi_S, |S| \geq 2} \exp\left(-\min_{t \in D^S} \phi_S(t)\right) \right)$$

Proof. By definition, we have that

$$Z = \sum_{t \in D^X} \left(\prod_{\phi_S, |S|<2} \exp\left(-\phi_S(t)\right) \cdot \prod_{\phi_S, |S| \geq 2} \exp\left(-\phi_S(t)\right) \right)$$

Trivially, $\exp\left(-\phi_S(t)\right) \leq \max_{t \in D^S} \left(\exp\left(-\phi_S(t)\right)\right) = \exp\left(-\min_{t \in D^S} \phi_S(t)\right)$ (by monotonicity). Applying this to the right term above, and exploiting the fact that this term now does not depend on t, we get that:

$$Z \leq \left(\sum_{t \in D^X} \prod_{\phi_S, |S|<2} \exp\left(-\phi_S(t)\right) \right) \cdot \left(\prod_{\phi_S, |S| \geq 2} \exp\left(-\min_{t \in D^S} \phi_S(t)\right) \right)$$

Since the set $\{\phi_S, |S| < 2\}$ contains only unary or constant functions, distributivity allows to swap sum and product and the result follows. Notice that this bound can be computed in linear time.   □

This bound can be strengthened by selecting a subset of all pairwise potentials in Φ defining a partial spanning k-tree $T \subset \Phi$. By applying a sum-product non serial dynamic programming [11] on $T' = T \cup \{\phi_S \in \Phi : |S| < 2\}$, we can obtain the exact $Z_{T'}$ for this sub-MRF in polynomial time. We can multiply $Z_{T'}$ by $(\prod_{\phi_S \in \Phi \setminus T'} \exp(-\min_{t \in D^S} \phi_S(t)))$ and get a tighter upper bound on Z which we denote Ub_T:

$$Z \leq Ub_T = \underbrace{\left(\sum_{t \in D^X} \prod_{\phi_S \in T'} \exp\left(-\phi_S(t)\right) \right)}_{\text{Computed using non serial dynamic programming}} \cdot \left(\prod_{\phi_S \in \Phi \setminus T'} \exp\left(-\min_{t \in D^S} \phi_S(t)\right) \right)$$

Proof. The proof is essentially similar to the previous one, and obtained by just replacing the set $\{\phi_S, |S| < 2\}$ and its complement set $\{\phi_S, |S| \geq 2\}$, defining the ranges of the products by the sets $\{\phi_S \in T'\}$ (and its complement respectively). The first item can be simply computed in $O(nd^2)$ time using non serial dynamic programming. □

These bounds alone are very weak. To further strengthen them, we reformulate the MRF using soft arc-consistencies [9] on its energy representation [1]. Soft arc consistencies essentially shift energy from pairwise potential functions to unary potential functions and eventually to the constant potential function $\phi_\varnothing$ while preserving equivalence. The result of this is an equivalent MRF (defining the same distribution) with increased unary and constant $\phi_\varnothing$ potential functions and pairwise potential functions that satisfy $\min_{t \in D^s} \phi_S(t) = 0$. Besides strengthening the bounds, it removes the need to compute the right term which is always equal to 1. Ub_1 and Ub_T can then be computed in $O(nd)$ and $O(nd^2)$ instead of $O(ed^2)$ (extension to non pairwise potentials would require the use of partial k-trees instead of trees and change the d^2 into a d^{k+1}).

In the rest of the paper we consider spanning trees and try both Existential Directional Arc Consistency (EDAC) and Virtual Arc Consistency (VAC) [9] as possible ways of strengthening Ub_1 and Ub_T.

4   Experimental Evaluation and Comparison

To evaluate the ability of the Z_ε^* algorithm to provide guaranteed deterministic approximations to Z, we implemented it on the top of the open source `toulbar2` solver[2]. The variable and value ordering used are the default weighted-degree and last conflict variable ordering and the existential support value-ordering [9]. We enforce EDAC at the root node and during search as usual for optimization. When VAC is used, it is only enforced at the root node because of its computational cost. Instead of the k-way branching described in Algorithm 1, we use a binary branching that either includes or reject a chosen value a at each branching decision. At each node, all variables of degree ≤ 2 are eliminated. The upper bound Ub_T uses a fixed maximum spanning tree with maximum sum of mean cost after enforcing arc consistencies at the root node. Our implementation is limited to pairwise potentials.

We compared it to different exact weighted counting approaches in terms of efficiency and quality of our guaranteed approximation. Four different exact counters have been considered. The first one is the already described `vec` exact counter [11]. The second one is the exact SAT based weighted counting tool `cachet` [33][3]. `cachet` relies internally on the Zchaff SAT solver to enumerate models with non zero weight and uses context-sensitive independence to cache intermediate counts. We also used the `ace` 3.0 compiler [8], using the

---

UAI competition executable provided in the `ace` distribution (always using a pseudo-random generator seed of 0). `ace` computes a tree decomposition and based on the obtained width may either perform tabular variable elimination or encode to CNF and compile in d-DNNF using `c2d`. We also tested the recent `minic2d` Sentential Decision Diagram (SDD) compilation package [28]. SDD are more constrained than d-DNNF and may therefore lead to larger compiled forms than d-DNNF, but since we do not need a compiled form and just the value of Z, we used the `-W` option of `minic2d` that performs weighted counting without compilation hoping to trade space for time. `minic2d` relies on its own internal SAT solver which is provided as a compiled binary in the distributed `minic2d` package. Because some of the compared solvers (`vec`, `cachet`) provide only a double floating representation of Z (or its logarithm), all software has been used in double floating point mode.

All executions have been performed on one core of an Intel® Xeon® CPU E5-2680 v3 @ 2.50 GHz (a Q4 2014 cpu) with a limit of 60 GB on RAM usage.

4.1   MRF to #SAT Encoding

If `ace` uses its own internal optimized MRF to SAT encoding, both `cachet` and `minic2d` require specific SAT encoding. Exact #SAT weighted counters use weighted literals and define the weight of a model as the product of the weights of all literals which are true in the model. They therefore rely on multiplicative potentials $\exp(-\phi_S(t))$. To transform an MRF into a literal-weighted CNF formula with a weighted count equal to the partition function, we use the ENC1 encoding of [8], originally described in [10]. This encoding is the CNF version of the so-called *local polytope*-based ILP encoding introduced in [34] for MRFs and [21] for weighted CSPs [9]. For each variable $i \in X$, we use one proposition $d_{i,r}$ for each value $r \in D^i$. This proposition is true iff variable i is assigned the value r. We encode *At Most One* (AMO) with hard clauses $(\neg d_{i,r} \lor \neg d_{i,s})$ for all $i \in X$ and all $r < s$, $r, s \in D^i$, as well as *At Least One* (f) with one hard clause $(\bigvee_r d_{i,r})$ for each i. These clauses ensure that the propositional encoding allows exactly one value for each variable in each model. For each potential ϕ_S, and each tuple $t \in D^S$, we have a propositional variable $p_{S,t}$. For non-zero energies $\phi_S(t)$, we have the literal $p_{S,t}$ with weight $\exp(-\phi_S(t))$. This represents the multiplicative potential to use if the tuple t is used. $\neg p_{S,t}$ is instead weighted by 1, the identity for multiplication. For every variable $i \in S$, we have a hard clause $(d_{i,t[i]} \lor \neg p_{S,t})$. These clauses enforce that if tuple t is used, its values $t[i]$ must be used. Then, for each variable $i \in S$ and each value $r \in D^i$, we have hard clauses $(\neg d_{i,r} \lor \bigvee_{t \in D^S, t[i]=r} p_{S,t})$ that enforces that if a value $r \in D^i$ is used, one of the allowed tuples $t \in D^S$ such that $t[i] = r$, $w_S(t) < k$ must be used.

It is interesting to notice that for pure Constraint Satisfaction Problems (MRFs having only $0/\infty$ potentials), it is known that Unit Propagation (UP) on this encoding enforces arc consistency in the original CSP [3].

We apply obvious optimization steps, explicitly forbidding local assignments with zero mass (sources of determinism). This encoding can be directly fed into

`minic2d`. Large problems however could not be encoded because `minic2d` only allows to express weights in a one-line list of maximum 100,000 chars in length[4].

In `cachet`, weighted literals l are either such that l and $\bar{l}$ receive a mass of 1 that has no effect on final mass, or such that the weights of a variable and its negation sum to 1. This is sufficient and convenient to express Bayesian nets because of their local normalization constraint. For arbitrary MRFs, for every $p_{S,t}$ corresponding to a mass $m = \exp(-\phi_S(t))$ we introduce another propositional variable $n_{S,t}$ with weights m (positive) and $1 - m$ (negative) and a simple implication clause $p_{S,t} \rightarrow n_{S,t}$. This extra variable is connected to the rest of the problem only through this clause and can therefore easily be eliminated, leading to a multiplicative factor m in models where $p_{S,t}$ is true and $1 = m + (1 - m)$ in models where $p_{S,t}$ is false, as required.

4.2   Benchmarks

Two types of benchmarks have been used. The first type of benchmark is made of instances of partition function computation appearing as sub-problems of binding affinity computations on molecular systems defined by a protein interacting
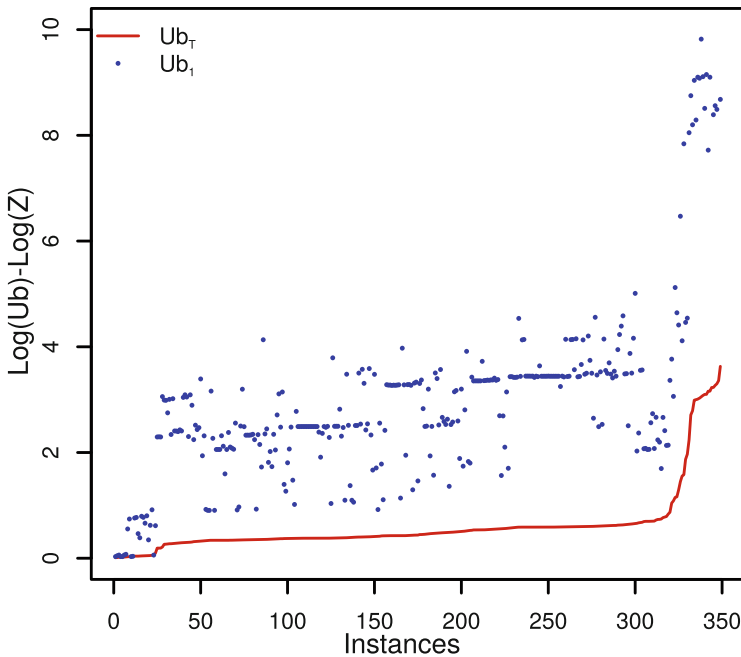

Fig. 2. Gap to Z: we represent $\log(Ub) - \log(\hat{Z})$ at the root node for Ub_1 and Ub_T using EDAC tightening (only negligible difference with VAC on these problems). The instances on the x axis are sorted in increasing gap size for the strongest Ub_T bound.

---

[4] This parameter could not be changed, being in the non open-source part on `minic2d`.
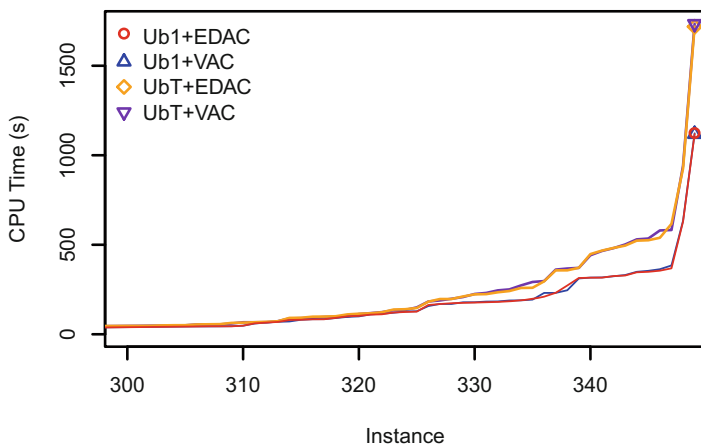
Fig. 3. Cactus plot of the running times of Z_ε^* with our four bounds. A point at (x, y) indicates that the number of solved problems is x if a deadline of y seconds is imposed on each resolution.
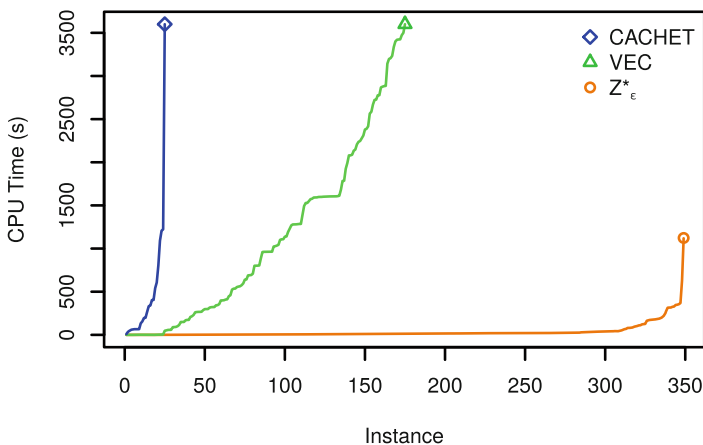

Fig. 4. Cactus plot of the running times of Z_ε^* with the Ub_1+EDAC bound together with `cachet` and `vec`. A point at (x, y) indicates that the number of solved problems is x if a deadline of y seconds is imposed on each resolution.

with a peptide or an amino-acid. The 3D model of these molecular systems were derived from crystallographic structures of the proteins in complex with their ligands, deposited in the Protein Data Bank. Missing heavy atoms in crystal structures as well as hydrogen atoms were added using the *tleap* module of the *Amber 14* software package [6]. The molecular all-atom *ff14SB* force field was used for the proteins and the ligands (peptides and amino-acids). The molecular systems were then subjected to 1000 steps of energy minimization with the *Sander* module of *Amber 14*. Next, a portion of the proteins including amino-

acids at the interface between the protein and the ligand as well as surrounding amino-acids with at least one atom within 8 to 12 Å (according to the molecular system) of the interface was selected.[5]

To evaluate the effect of the strength of the upper bound on the algorithm efficiency, we applied Z_ε^* with $\varepsilon = 10^{-3}$ on a series of 349 systems using the four

Table 1. Time results for UAI/PIC'2011 instances. Three different categories are represented: Boltzmann machines (rbm) with attractive (ferro) and non attractive coupling, Grids, and graph problems. Running-times are given in seconds. M: Memory Out (60 GB), T: Time out (1 h). Bold is best.

Instance	Z_ε^*	minic2d	ace	vec	cachet
smokers_10	**< 0.01**	0.663	**< 0.01**	**< 0.01**	0.264
smokers_20	**< 0.01**	312.825	**< 0.01**	**< 0.01**	332.168
rbm_20	7.46	T	**3.376**	16.17	941.14
rbm_ferro_20	**< 0.01**	T	3.24	16.18	893.84
rbm_21	13.75	T	**6.854**	34.35	2041.64
rbm_ferro_21	**< 0.01**	T	6.868	34.17	1975.78
rbm_22	33.75	T	**14.411**	72.78	T
rbm_ferro_22	**< 0.01**	T	14.418	72.43	T
grid10x10.f10	66.32	**< 0.01**	**< 0.01**	**< 0.01**	T
grid20x20.f10	T	T	**13.316**	2104.57	T
grid20x20.f15	T	T	**13.665**	2099.24	T
grid20x20.f2	T	T	**13.603**	2107.92	T
grid20x20.f5	T	T	**13.609**	2102.32	T
GEOM30a_3	**< 0.01**	**< 0.01**	**< 0.01**	**< 0.01**	2.668
GEOM30a_4	7.66	78.604	**< 0.01**	**< 0.01**	43.77
GEOM30a_5	67.48	368.361	**< 0.01**	**< 0.01**	405.38
GEOM40_2	**< 0.01**	**< 0.01**	**< 0.01**	**< 0.01**	**< 0.01**
GEOM40_3	**< 0.01**	**< 0.01**	**< 0.01**	**< 0.01**	3.62
GEOM40_4	1.42	0.58	**< 0.01**	**< 0.01**	616.59
GEOM40_5	12.67	12.801	**< 0.01**	**< 0.01**	828.50
myciel5g_3	**37.2**	T	M	T	T
myciel5g_4	M	T	M	T	T
myciel5g_5	M	T	M	T	T
queen5_5_3	367.2	T	**83.004**	945.20	T
queen5_5_4	**2423.67**	T	M	T	T

---

[5] Each of these systems requires extensive molecular modeling expertise to be properly defined. We intend to make this benchmark together with the Z_ε^* implementation available.

different bounds. For each system, the most complex partition function, defined on the compound system, is computed. The largest problem has 22 variables and the largest domain size is 34. The gap between our two bounds and the guaranteed approximation of Z determined by Z_ε^* is shown in Fig. 2. The bound Ub_T is clearly stronger than Ub_1, as expected.

We then compare the running times of Z_ε^* using these 4 bounds in a cactus plot in Fig. 3. The best bound in terms of run-time is the lightest Ub_1+EDAC bound confirming that stronger, thus more expensive, bounds may quickly become counter productive.

We represent the same information with the fastest Ub_1+EDAC and two of the three exact counting tools in Fig. 4. We omit `ace` and `minic2d`. Indeed, `ace` was able to solve only 17 problems within the time limit and failed on all remaining problems with a memory exception (despite the explicit allocation of 60 GB to the JAVA machine). `minic2d` was instead unable to model 294 systems out of the 349 because of its previously mentioned limitation on the length of the weight line. On the remaining 55 problems, `minic2d` solved 7 problems in less than one hour.

In the rest of the experiments we therefore use the Ub_1+EDAC upper bound which seems the most efficient bound. To see how the Z_ε^* algorithm performs on other types of problems, we used instances extracted from UAI and PIC'2011 challenge instances (PR task)[6] that use only pairwise potentials as a second set of benchmark. Using the same value of $\varepsilon = 10^{-3}$, we again compared Z_ε^* with `vec`, `cachet`, `ace` and `minic2d`.

The results clearly show there is no single winner: except for `cachet` which is always dominated by one of the other solvers, each algorithm may outperform others. Specifically, the Z_ε^* algorithm, despite its lack of sophisticated caching technology, is able to outperform its competitors in various cases. Nevertheless, `minic2d` outperformed Z_ε^* on the Grid category (probably because of the combination of Boolean variables and relatively small treewidth), itself outperformed by `vec` and further outperformed by `ace` (Table 1).

5   Conclusion

Existing solvers providing deterministic guarantees for partition function computation exploit two sources of efficiency. This first one is caching of local counts based on context-sensitive independence [33], related to tree-decomposition. The other one is determinism *i.e.,* the existence of zero probability assignments allowing to prune zero probability mass sub-trees during search. This second source of efficiency will provide significant speedups only when a significant fraction of the search space has 0 probability. Such distributions have very low entropy.

In this paper, motivated by the computation of statistical estimate of affinity between bio-molecules, we have proposed to build upon existing optimization technology to provide a new source of pruning for partition function computation with deterministic guarantees: a branch and bound-based schema equipped

---

[6] http://www.cs.huji.ac.il/project/PASCAL.

with upper bounds derived from soft local consistencies. As existing SAT-based exact approaches, our algorithm exploits determinism and a much simpler and less powerful form of caching than those based on tree-decomposition. It is however able to prune regions of proven negligible mass of probability and is therefore able to exploit relatively low entropy distributions having a much wider support, including those with no determinism. The resulting algorithm offers an adjustable deterministic guarantee on the quality of the computed partition function and, despite its limited caching strategy, may already offer interesting speedups compared to exact solvers.

Z_ε^* includes two crucial ingredients to quickly gather large number of probability masses: pruning based on very fast incremental upper bounds derived from optimization bounds and on-the-fly sum-prod elimination. An important point is that these ingredients can be easily injected into existing SAT-based counters, including knowledge-compilation based counters using SAT-solver traces. This could be achieved by defining counting upper bounds from existing Max-SAT bounds. These bounds have already been related to soft arc-consistency bounds [4,23,24]. This should extend their range of application to guaranteed approximate probabilistic inference on problems with limited or no determinism.

From an affinity computation point of view, the next step is now to evaluate the actual empirical quality of the association constant estimation provided by the computed ratio of partition functions. Beyond algorithmic approximations, the modeling may also have important effects on the estimated value based on different rotamer discretizations, relative positions of molecules in the complex or weights of different contributions in the energy function. To pursue this target, we intend to use available databases that provide experimental values of the association constant of various protein-ligand complexes following various mutations on one of the partners. To keep the modeling to a reasonable level of complexity, this will be preferably achieved on protein-protein complexes.

References

1. Allouche, D., André, I., Barbe, S., Davies, J., de Givry, S., Katsirelos, G., O'Sullivan, B., Prestwich, S., Schiex, T., Traoré, S.: Computational protein design as an optimization problem. Artif. Intell. **212**, 59–79 (2014)
2. Allouche, D., de Givry, S., Katsirelos, G., Schiex, T., Zytnicki, M.: Anytime hybrid best-first search with tree decomposition for weighted CSP. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 12–29. Springer, Heidelberg (2015)
3. Bacchus, F.: GAC via unit propagation. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 133–147. Springer, Heidelberg (2007)
4. Bonet, M.L., Levy, J., Manyà, F.: Resolution for max-sat. Artif. Intell. **171**(8), 606–618 (2007)

5. Campeotto, F., Dal Palu, A., Dovier, A., Fioretto, F., Pontelli, E.: A constraint solver for flexible protein model. J. Artif. Intell. Res. **48**, 953–1000 (2013)
6. Case, D., Babin, V., Berryman, J., Betz, R., Cai, Q., Cerutti, D., Cheatham III, T., Darden, T., Duke, R., Gohlke, H., et al.: Amber 14 (2014)
7. Chakraborty, S., Fremont, D.J., Meel, K.S., Seshia, S.A., Vardi, M.Y.: Distribution-aware sampling and weighted model counting for SAT. In: Proceedings of the 28th Conference on Artificial Intelligence, pp. 1722–1730 (2014)
8. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artif. Intell. **172**(6), 772–799 (2008)
9. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artif. Intell. **174**, 449–478 (2010)
10. Darwiche, A.: A logical approach to factoring belief networks. In: KR 2002, pp. 409–420 (2002)
11. Dechter, R.: Bucket elimination: a unifying framework for reasoning. Artif. Intell. **113**(1–2), 41–85 (1999)
12. Dechter, R.: Reasoning with probabilistic and deterministic graphical models: Exact algorithms. Synth. Lect. Artif. Intell. Mach. Learn. **7**(3), 1–191 (2013)
13. Dechter, R., Flerova, N., Marinescu, R.: Search algorithms for m best solutions for graphical models. In: AAAI. Citeseer (2012)
14. Ermon, S., Gomes, C.P., Sabharwal, A., Selman, B.: Taming the curse of dimensionality: Discrete integration by hashing and optimization. arXiv preprint arXiv:1302.6677 (2013)
15. Georgiev, I., Lilien, R.H., Donald, B.R.: The minimized dead-end elimination criterion and its application to protein redesign in a hybrid scoring and search algorithm for computing partition functions over molecular ensembles. J. Comput. Chem. **29**(10), 1527–1542 (2008)
16. Gilks, W.R.: Markov Chain Monte Carlo. Wiley Online Library (2005)
17. Jaakkola, T.S.: Tutorial on variational approximation methods. In: Advanced Mean Field Methods: Theory Practice, p. 129 (2001)
18. Karanicolas, J., Kuhlman, B.: Computational design of affinity and specificity at protein-protein interfaces. Curr. Opin. Struct. Biol. **19**(4), 458–463 (2009)
19. King, N.P., Bale, J.B., Sheffler, W., McNamara, D.E., Gonen, S., Gonen, T., Yeates, T.O., Baker, D.: Accurate design of co-assembling multi-component protein nanomaterials. Nature **510**(7503), 103–108 (2014)
20. Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artif. Intell. **6**(4), 293–326 (1976)
21. Koster, A.: Frequency assignment: Models and Algorithms. Ph.D. thesis, University of Maastricht, The Netherlands (1999). www.zib.de/koster/thesis.html
22. Larrosa, J.: Boosting search with variable elimination. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 291–305. Springer, Heidelberg (2000)
23. Larrosa, J., Heras, F.: Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In: Proceedings of the 19th IJCAI, Edinburgh, Scotland, pp. 193–198 (2005)
24. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-sat solving. Artif. Intell. **172**(2–3), 204–233 (2008)
25. Liu, Q., Ihler, A.T.: Bounding the partition function using holder's inequality. In: Proceedings of the 28th International Conference on Machine Learning (ICML 2011), pp. 849–856 (2011)
26. Lovell, S.C., Word, J.M., Richardson, J.S., Richardson, D.C.: The penultimate rotamer library. Proteins **40**(3), 389–408 (2000). http://www.ncbi.nlm.nih.gov/pubmed/10861930

27. Miklos, A.E., Kluwe, C., Der, B.S., Pai, S., Sircar, A., Hughes, R.A., Berrondo, M., Xu, J., Codrea, V., Buckley, P.E., et al.: Structure-based design of supercharged, highly thermoresistant antibodies. Chem. Biol. **19**(4), 449–455 (2012)
28. Oztok, U., Darwiche, A.: A top-down compiler for sentential decision diagrams. In: Proceedings of the 24th International Conference on Artificial Intelligence. AAAI Press (2015)
29. Pearlman, D.A., Case, D.A., Caldwell, J.W., Ross, W.S., Cheatham, T.E., DeBolt, S., Ferguson, D., Seibel, G., Kollman, P.: Amber, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. Comput. Phys. Commun. **91**(1), 1–41 (1995)
30. Roth, D.: On the hardness of approximate reasoning. Artif. Intell. **82**(1), 273–302 (1996)
31. Röthlisberger, D., Khersonsky, O., Wollacott, A.M., Jiang, L., DeChancie, J., Betker, J., Gallaher, J.L., Althoff, E.A., Zanghellini, A., Dym, O., Albeck, S., Houk, K.N., Tawfik, D.S., Baker, D.: Kemp elimination catalysts by computational enzyme design. Nature **453**(7192), 190–195 (2008). http://www.ncbi.nlm.nih.gov/pubmed/18354394
32. Sammond, D.W., Eletr, Z.M., Purbeck, C., Kuhlman, B.: Computational design of second-site suppressor mutations at protein-protein interfaces. Prot. Struct. Funct. Bioinform. **78**(4), 1055–1065 (2010)
33. Sang, T., Beame, P., Kautz, H.: Solving bayesian networks by weighted model counting. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005). vol. 1, pp. 475–482 (2005)
34. Schlesinger, M.: Syntactic analysis of two-dimensional visual signals in noisy conditions. Kibernetika **4**, 113–130 (1976)
35. Silver, N.W., King, B.M., Nalam, M.N., Cao, H., Ali, A., Kiran Kumar Reddy, G., Rana, T.M., Schiffer, C.A., Tidor, B.: Efficient computation of small-molecule configurational binding entropy and free energy changes by ensemble enumeration. J. Chem. Theory Comput. **9**(11), 5098–5115 (2013)
36. Simoncini, D., Allouche, D., de Givry, S., Delmas, C., Barbe, S., Schiex, T.: Guaranteed discrete energy optimization on large protein design problems. J. Chem. Theory Comput. **11**(12), 5980–5989 (2015)
37. Toda, S.: On the computational power of PP and ⊕P. In: 30th Annual Symposium on Foundations of Computer Science, pp. 514–519. IEEE (1989)
38. Valiant, L.G.: The complexity of computing the permanent. Theoret. Comput. Sci. **8**(2), 189–201 (1979)
39. Viricel, C., Simoncini, D., Allouche, D., de Givry, S., Barbe, S., Schiex, T.: Approximate counting with deterministic guarantees for affinity computation. In: Le Thi, H.A., Dinh, T.P., Nguyen, N.T. (eds.) Model. Comput. & Optim. in Inf. Syst. & Manage. Sci. AISC, vol. 360, pp. 165–176. Springer, Switzerland (2015)
40. Wainwright, M.J., Jaakkola, T.S., Willsky, A.S.: A new class of upper bounds on the log partition function. IEEE Trans. Inf. Theory **51**(7), 2313–2335 (2005)

Music Track

Finding Alternative Musical Scales

J.N. Hooker$^{(\boxtimes)}$

Carnegie Mellon University, Pittsburgh, USA
`jh38@andrew.cmu.edu`

Abstract. We search for alternative musical scales that share the main advantages of classical scales: pitch frequencies that bear simple ratios to each other, and multiple keys based on an underlying chromatic scale with tempered tuning. We conduct the search by formulating a constraint satisfaction problem that is well suited for solution by constraint programming. We find that certain 11-note scales on a 19-note chromatic stand out as superior to all others. These scales enjoy harmonic and structural possibilities that go significantly beyond what is available in classical scales and therefore provide a possible medium for innovative musical composition.

1 Introduction

The classical major and minor scales of Western music have two characteristics that make them a fertile medium for musical composition: pitch frequencies that bear simple ratios to each other, and multiple keys based on an underlying chromatic scale with tempered tuning. Simple ratios allow for rich and intelligible harmonies, while multiple keys greatly expand possibilities for complex musical structure. While these traditional scales have provided the basis for a fabulous outpouring of musical creativity, expressive power, and structural sophistication over several centuries, one might ask whether alternative scales with the same favorable characteristics—simple ratios and multiple keys—could unleash even greater creativity.

We take a step toward answering this question by undertaking a systematic search for musically appealing alternative scales. We restrict ourselves to diatonic scales, whose adjacent notes are separated by a whole tone or semitone. We conduct the search by defining a constraint satisfaction model that, for each suitable diatonic scale, seeks to assign relatively simple ratios to intervals in the scale. The ratios must be amenable to tuning based on equal temperament. To the extent that such an assignment of ratios is possible, the scale is a potential candidate for musical use.

Constraint programming is well adapted to this problem because it naturally expresses a recursive condition requiring that each note bear simple ratios with some other notes, but not necessarily with the tonic. The constraint programming model also solves in a reasonable amount of time.

We find that while the classical 7-note scales deserve the attention they have received, certain 11-note scales based on a 19-note chromatic stand out

as possibly even more attractive, based on the criteria developed here. To our knowledge, this is a new result.

After a brief review of previous work, we present a rationale for preferring the simple ratios and multiple keys that characterize the classical scales. We then discuss how these characteristics can be precisely formulated as criteria for nonstandard scales, and we state a constraint programming model that formulates the criteria. We then present computational results, focusing on scales based on the 12-note and 19-note chromatics. We exhibit several particular scales that composers may wish to investigate.

2   Previous Work

Composers have experimented with a number of alternative scales in recent decades. One of the most discussed is the Bohlen-Pierce scale, which consists of 9 notes on a 13-tone tempered chromatic scale [4,9]. The scale spans a twelfth, rather than the traditional octave. It treats notes that lie a twelfth apart as equivalent, much as traditional scales treat notes an octave apart as equivalent. Composers Richard Boulanger, Ami Radunskaya and Jon Appleton have written pieces using the Bohlen-Pierce scale [18]. In this paper we focus instead on scales that span the traditional octave, due to the ear's strong tendency to identify tones an octave apart, and the interesting possibilities that remain to be explored among these scales. Pierce [17] also experimented with a scale that divides the octave into 8 equal intervals, but we will find this scale to be unappealing due to the lack of simple pitch ratios.

A number of composers have written music that uses the quarter-tone scale, in which the octave is divided into 24 equal intervals. Some of the better-known examples are Béla Bartók, Alban Berg, Ernest Bloch, Pierre Boulez, Aaron Copeland, George Enescu, Charles Ives, and Henry Mancini. We will find, however, that quarter tones do not offer significant musical advantages, at least according to the criteria developed here.

Benson [2] reports that several composers have experimented with "super just" scales that use only perfect ratios. These include Harry Partch (43-tone scale), Wendy Carlos (12 tones), Lou Harrison (16 tones), Wildred Perret (19 tones) [16], John Chalmers (a similar 19-tone scale), and Michael Harrison (24 tones). These scales are not fitted to a tempered chromatic scale as are the scales we discuss here and therefore lack the advantage of providing multiple keys.

Combinatorial properties of scales, keys and tonality have been studied by Balzano [1], Noll [11–13], and others [6,7,15,22]. The composer Olivier Messiaen studied "modes of limited transposition" (scales with fewer keys than notes in the underlying chromatic) [10].

Sethares [20] formulates an optimization problem for finding an instrumental timbre (i.e. relative strength of upper harmonics) that maximizes the degree to which the notes of a given scale sound consonant with the tonic when played on that instrument. The object is to design an instrument that is most suitable for a given scale, rather than to find possible scales.

To our knowledge, no previous study conducts a systematic search for scales with simple pitch ratios and multiple keys, nor formulates an optimization or constraint satisfaction problem for conducting such a search.

3   Characteristics of Standard Scales

We first provide a rationale for preserving the main characteristics of standard scales: intervals that correspond to simple frequency ratios, and multiple keys based on tempered tuning.

3.1   Simple Ratios

A *harmonic partial* of a tone (or a harmonic, for short) is an equal or higher tone whose frequency is an integral multiple of the frequency of the original tone. Two tones whose frequencies bear a simple ratio have many harmonics in common, and this helps the ear to recognize the interval between the tones. If the frequency ratio is a/b (where $a > b$ and a, b are coprime), every ath harmonic of the lower tone coincides with every bth harmonic of the upper tone. For example, if $a/b = 3/2$ as in a perfect fifth, every third harmonic of the lower tone coincides with every other harmonic of the upper tone. This coincidence of harmonics is aurally important because a tone produced by almost any acoustic instrument is accompanied by many upper harmonics (or perhaps only odd harmonics, as in the case of a clarinet). The ear therefore learns to associate a given interval with the timbre produced by a certain coincidence of harmonics, and this distinctive timbre makes the interval easier to recognize. In particular, the octave interval tends to be perceived as a unison, because the upper note adds nothing to the harmonic series: every harmonic of the upper note is a harmonic of the lower.

This ease of recognition benefits both harmony and counterpoint, which might be viewed as the two principal mechanisms of Western polyphonic music. The benefit to harmony is clear. It is hard to distinguish one tone cluster from another if the pitch frequencies have no discernible ratios with each other, while if the ratios are simple, a given tone cluster generates a series of harmonics that reinforce each other in a recognizable pattern. Harmony can scarcely play a central role in music if listeners cannot distinguish which chord they are hearing. In addition, harmony adds immeasurably to the composer's expressive palette. Because each chord has its own peculiar timbre, shifting from one set of frequency ratios to another can create a wide variety of effects the listener can readily appreciate, as does moving from 4:5:6 to 10:12:15 (major to minor triad) or from 8:10:12:15 to 12:15:18:20 (major seventh to a "softer" major sixth chord). The expressive use of harmony has been a key element of music at least since J.S. Bach and became especially important for impressionist and jazz composers.[1]

---

[1] Simple ratios also tend to produce intervals that are consonant in some sense, although consonance and dissonance involve other factors as well. One theory is that the perception of dissonance results from beats that are generated by upper

Recognizable intervals are equally important for counterpoint, because without them, simultaneous moving voices are perceived as cacophony. Voices that create recognizable harmonic relationships, on the other hand, can be perceived as passing tones from one recognizable chord to another, thus making counterpoint intelligible. This is confirmed by Schenkerian analysis, which interprets Western music as consisting largely of underlying major and minor triads connected by passing tones [5,14].

3.2   Multiple Keys

Multiple keys enable a signature trait of Western musical structure: the ability to begin in a tonic key, venture away from the tonic into exotic keys, and eventually return "home" to the tonic with an experience of satisfaction and closure. Multiple keys are implemented by embedding the corresponding 7-note scales within a single 12-note "chromatic" scale with tempered tuning. For example, one can play a major scale rooted at any tone of the chromatic scale by sounding the 1st, 3rd, 5th, 6th, 8th, 10th, and 12th notes of the chromatic scale beginning at that tone. This results in 12 distinct major keys.

It is remarkable that the frequency ratios that define classical scales are closely matched by the pitches in a tempered chromatic scale. The pitches are "tempered" in the sense that they are adjusted so that no key is too far out of tune. Various types of temperament have been used historically, but the modern solution is to use equal temperament, in which the kth pitch of the chromatic scale has a frequency ratio of $2^{(k-1)/12}$ with the first pitch. Table 1 shows tuning errors that result for the major diatonic scale. For example, the fifth note of the scale is slightly flat when played on a tempered scale, and the third note is sharp. None of the errors is greater than 0.9 %, or about 16 cents.[2]

Temperament was originally adopted to allow a musical instrument with fixed tuning (such as a piano or organ) to play in all keys. But it has an equally important function in musical composition. It allows one to move into a different key by changing only a few notes of the tonic key, where more "distant" keys share fewer notes with the tonic. For example, the most closely related keys, the dominant and subdominant (rooted at the fifth and fourth note) share 6 of the 7 notes of the tonic key. This allows the composer to exploit a wide range of possible relationships when moving from one key to another, making the musical texture richer and more interesting.

---

harmonics that are close in frequency [18–21]. We will occasionally refer to simple ratios as resulting in "consonant" intervals, but this is not to deny the other factors involved.

[2] We use the tempered pitch as a base for the percentage error because it is the same across all scales and so permits more direct comparison of errors. A cent is 1/1200 of an octave, or 1/100 of a semitone. Thus if two tones differ by c cents, the ratio of their frequencies is $2^{c/1200}$. An error of $+0.9$ % is equivalent to $+15.65$ cents, and an error of -0.9% to -15.51 cents.

Table 1. Relative pitch errors of the equally tempered major diatonic scale, as a percentage of tempered tuning. Positive errors indicate sharp tuning, negative errors flat tuning.

Note	Perfect ratio	Tempered ratio	Error %	Error cents
1	1/1	1.00000	0.000	0
2	9/8	1.12246	−0.226	−3.91
3	5/4	1.25992	+0.787	+13.69
4	4/3	1.33484	+0.113	+1.96
5	3/2	1.49831	−0.113	−1.96
6	5/3	1.68179	+0.899	+15.64
7	15/8	1.88775	+0.675	+11.73

4   Requirements for Alternative Scales

Given the advantages of simple ratios and multiple keys, we will attempt to generate alternative scales with these same characteristics. In general, a scale will have m notes on a chromatic scale of n notes. The equally tempered chromatic pitches should result in intervals with something close to simple ratios.

4.1   Keys and Temperament

The first decision to be made is the tolerance for inaccurate tuning in the tempered scale. The only reliable guide we have is two centuries of experience with the equally tempered 12-tone chromatic. It is famous for producing flat fifths, but the error is much greater for major thirds and sixths, which are sharp. The tempered major third is in fact quite harsh, although we have learned to tolerate it, and the error is magnified in the upper partials. It therefore seems prudent to limit the relative error to the maximum error in the traditional major scale, namely ±0.9 %, or between −15.51 and +15.65 cents.

There are $\binom{n}{m}$ scales of m notes on n chromatic pitches, but many of these scales are aesthetically undesirable. We can begin by considering only diatonic scales, whose adjacent notes are no more than two chromatic tones (semitones) apart. Diatonic scales are easier to perform, and restricting ourselves to them helps keep the complexity of the search within bounds.[3]

A diatonic scale can be represented by a binary tuple $s = (s_1, \ldots s_m)$, where $s_i + 1$ is the number of semitones between note i and note $i + 1$. Because there are n semitones altogether, s must contain $m_0 = 2m - n$ zeros and $m_1 = n - m$ ones. This means that there are $\binom{m}{m_0} = \binom{m}{m_1}$ diatonic scales to consider.

We also adopt the aesthetic convention that semitones should be distributed fairly evenly through the scale rather than bunched up together. One approach is

---

[3] This restriction excludes the classical harmonic minor scale, in which notes 6 and 7 are separated by three semitones, but the harmonic minor scale can be viewed as a variant of a natural minor scale in which note 7 is raised a semitone for cadences.

to require the scales to have Myhill's property, discussed by Noll [12]. However, because few scales satisfy this strong property, we require that the scales have a minimum number of semitone and whole-tone adjacencies. That is, the number of pairs (s_i, s_{i+1}) in which $s_i = s_{i+1}$ should be minimized subject to the given m and n, where s_{m+1} is cyclically identified with s_1. If $m_0 \geq m_1$, the number k_0 of pairs of adjacent zeros can be as few as $m_0 - m_1$, and the number k_1 of adjacent ones can be zero. The reasoning is similar if $m_1 \geq m_0$. We therefore require

$$k_0 = m_0 - \min\{m_0, m_1\}, \quad k_1 = m_1 - \min\{m_0, m_1\}$$

It is not hard to show that the number of diatonic scales satisfying this requirement is

$$\binom{\max\{m_0, m_1\}}{\min\{m_0, m_1\}} + \binom{\max\{m_0, m_1\} - 1}{\min\{m_0, m_1\} - 1} \tag{1}$$

For example, among 7-note scales on a 12-note chromatic, we have $(m_0, m_1) = (2, 5)$, $(k_0, k_1) = (0, 3)$, and $\binom{5}{2} + \binom{4}{1} = 14$ suitable scales.

The number of keys generated by a given scale s depends on the presence of any cyclic repetition in s. Let Δ be the smallest offset that results in the same 0/1 pattern; that is, Δ is the smallest positive integer such that $s_i = s_{i+\Delta}$ for $i = 1, \ldots, m$, where $s_{m+1}, \ldots, s_{2m}$ are respectively identified with $s_1, \ldots, s_m$. Then there are

$$\Delta + \sum_{j=1}^{\Delta} s_j$$

distinct keys. For the classical major scale $s = (1, 1, 0, 1, 1, 1, 0)$, we have $\Delta = 7$, and there are $7 + \sum_{j=1}^{7} s_j = 12$ keys. When $\Delta < m$, we have a "mode of limited transposition" [10]. For example, the whole tone scale favored by Debussy has $s = (1, 1, 1, 1, 1, 1)$ and $\Delta = 1$, yielding only $1 + s_1 = 2$ keys, which have no notes in common.

4.2   Simple Ratios

In the previous section, we geneated scales by considering subsets of notes in a chromatic scale. For each such scale, we now wish to determine whether relatively simple ratios can be assigned to the notes of the scale that are within 0.9 % of the tempered pitches. It does not seem necessary that every note be consonant with the tonic, because many of the harmonies that occur in music do not involve the tonic. Yet every note should at least be consonant with another note of the scale, to allow it to take part in harmony at some point.

We therefore propose that possible ratios be obtained by *generators*, which are simple ratios that a given note can bear with some other note of the scale (these are not generators in the formal sense of group theory). Since we identify notes an octave apart, we consider notes in a two-octave range. Thus if $r_1, \ldots, r_p$ are the generators and f_i the frequency of note i, we require for each note

$i \in \{1, \ldots, m\}$ that

$$\frac{f_i}{f_j} = r_q \ \text{ or } \ \frac{2f_j}{f_i} = r_q \ \text{ or } \ \frac{f_j}{f_i} = r_q \ \text{ or } \ \frac{2f_i}{f_j} = r_q,$$
$$\text{for some } j \in \{1, \ldots, m\} \setminus \{i\}, \text{some } q \in \{1, \ldots, p\}$$

This requirement is insufficient, however, because it allows for subsets of notes that are consonant with others in the same subset but are extremely dissonant with notes in other subsets. To avoid this outcome, we make the requirement recursive, beginning with the tonic. That is, a note is acceptable if it bears a simple ratio with the tonic, or if it bears a simple ratio with another acceptable note. This can result in notes that are rather dissonant with the tonic, but they will always be consonant with notes that closely precede it in the recursion.

To express this in notation, we let $\pi = (\pi_1, \ldots, \pi_m)$ be a permutation of $1, \ldots, m$, where π_i is interpreted as the ith note defined in the recursion. Then frequencies $f_1, \ldots, f_m$ are acceptable if and only if $f_1 = 1$ and there is a permutation π with $\pi_1 = 1$ such that for each $i \in \{2, \ldots, m\}$, $1 < f_{\pi_i} < 2$ and

$$\frac{f_{\pi_i}}{f_{\pi_j}} = r_q \ \text{ or } \ \frac{2f_{\pi_j}}{f_{\pi_i}} = r_q \ \text{ or } \ \frac{f_{\pi_j}}{f_{\pi_i}} = r_q \ \text{ or } \ \frac{2f_{\pi_i}}{f_{\pi_j}} = r_q, \tag{2}$$
$$\text{for some } j \in \{1, \ldots, i-1\}, \text{some } q \in \{1, \ldots, p\}$$

Whenever $f_{\pi_i}/f_{\pi_j} = r_q$, we also have $2f_{\pi_j}/f_{\pi_i} = 2/r_q$. Thus there is no need to consider both r_q and $2/r_q$ as generators. That is, we need only consider reduced fractions with odd numerators. We will order the generators by decreasing simplicity, beginning with the smallest denominators, and for each denominator, beginning with the smallest numerator. The first several generators, in order of decreasing simplicity, are

$$\frac{3}{2}, \frac{5}{3}, \frac{5}{4}, \frac{7}{4}, \frac{7}{5}, \frac{9}{5}, \frac{7}{6}, \frac{11}{6}, \frac{9}{7}, \frac{11}{7}, \frac{13}{7}, \frac{9}{8}, \frac{11}{8}, \frac{13}{8}, \frac{15}{8}, \frac{11}{9}, \frac{13}{9}, \frac{17}{9} \tag{3}$$

5 Constraint Programming Model

Constraint programming is naturally suited to formulate the problem described above, because it readily accommodates the variable indices π_i that occur in the expressions f_{π_i} of the recursive formulation. To state the model, we write each frequency ratio f_i as a fraction a_i/b_i in lowest terms. We set $f_1 = 1$, so that f_i is the frequency ratio of note i with the tonic. In the model below, we treat π_i, a_i, and b_i as integer variables. Constraint (a) ensures that π is a permutation. Constraint (b) initializes the recursion. Constraint (c) requires that a_i/b_i be a valid ratio in lowest terms, where "coprime" is a pre-defined predicate. Constraint (d) reduces symmetry by requiring that ratios be indexed in increasing order. Constraints (e) and (f) enforce condition (2), where G is the set of generators. Constraint (g) requires that temperament lie within tolerance ϵ ($= 0.009$), where t_i indexes the chromatic tone corresponding to scale note i. Thus $t_1 = 1$ and $t_i = t_{i-1} + s_{i-1} + 1$ for $i = 2, \ldots, m$. The domains (h) place an

upper bound M on the denominators b_i, to limit the search and avoid intervals that are unreasonably dissonant.

$$
\begin{aligned}
&\text{alldiff}(\pi_1, \ldots, \pi_m) && (a)\\
&\pi_1 = a_1 = b_1 = 1 && (b)\\
&1 < \frac{a_i}{b_i} < 2, \ \ \text{coprime}(a_i, b_i), \ i = 1, \ldots, m && (c)\\
&\frac{a_{i-1}}{b_{i-1}} < \frac{a_i}{b_i}, \ i = 2, \ldots, m && (d)\\
&\bigvee_{j<i} \left[(\pi_i > \pi_j) \Rightarrow \left(\frac{a_{\pi_i}/b_{\pi_i}}{a_{\pi_j}/b_{\pi_j}} \in G \ \vee \ \frac{2a_{\pi_j}/b_{\pi_j}}{a_{\pi_i}/b_{\pi_i}} \in G \right) \right], \ i = 2, \ldots, m && (e)\\
&\bigvee_{j<i} \left[(\pi_i < \pi_j) \Rightarrow \left(\frac{a_{\pi_j}/b_{\pi_j}}{a_{\pi_i}/b_{\pi_i}} \in G \ \vee \ \frac{2a_{\pi_i}/b_{\pi_i}}{a_{\pi_j}/b_{\pi_j}} \in G \right) \right], \ i = 2, \ldots, m && (f)\\
&\frac{|a_i/b_i - 2^{(t_i-1)/n}|}{2^{(t_i-1)/n}} \leq \epsilon, \ i = 1, \ldots, m && (g)\\
&\pi_i \in \{1, \ldots, m\}, \ a_i \in \{1, \ldots, 2M\}, \ b_i \in \{1, \ldots, M\}, \ i = 1, \ldots, m && (h)
\end{aligned}
\tag{4}
$$

Conditions of the form $\alpha \in G$ in (e) and (f) are formulated by writing the constraint $\sum_{g \in G}(\alpha = g) \geq 1$. Fractions are shown in the constraints for readability, but they are removed in the model given to the solver, for example by writing $a/b < c/d$ as $ad < bc$.

6 Computational Results

The search algorithm was implemented in IBM OPL Studio 12.6.2. The OPL script language was used to search tempered scales s, for given values of n and m. The number of scales examined is given by (1). For each scale s, the model (4) was solved by the CP Optimizer to find acceptable ratios that are within tolerance of the tempered scale.

A key decision is what set G of generators to use. We found that in several cases, there were no solutions for the simplest generators. We therefore used a rather large set of generators for all scales, namely those in (3), which typically resulted in many solutions. Since the first solutions found tend to have the simplest ratios, we terminated the process after finding 50 solutions. Distinct solutions $(a_1/b_1, \ldots, a_m/b_m)$ were found by re-solving the problem with constraints that exclude the solutions already found. The solver generally obtained each solution in well under a minute, perhaps only a few seconds, depending on the number of chromatic tones. When the solver could no longer find a solution, it ran several hours without proving infeasibility. We therefore set the maximum computation time for finding each solution at 5 min, on the assumption that this suffices to find any remaining solution if it exists.

We focused on solutions $(a_1/b_1, \ldots, a_m/b_m)$ that can be obtained from relatively small generators. Since a given solution can typically be obtained from several distinct sets of generators, we computed for each solution and each scale note i the simplest generator that could derive it from another note. That is,

we computed for each note i the simplest of the following ratios that fall in the range [1,2], over all $j \neq i$:

$$\frac{a_i/b_i}{a_j/b_j}, \quad \frac{2a_j/b_j}{a_i/b_i}, \quad \frac{a_j/b_j}{a_i/b_i}, \quad \frac{2a_i/b_i}{a_j/b_j}$$

where simplicity is measured by the size of the denominator when the fraction is in reduced terms. We will call this resulting ratio the *minimal generator* for note i. The minimal generator need not be among the generators actually used to obtain the scale in the solution of the constraint programming model (4).

Each solution obtained for a given scale s represents one way the ear might interpret the frequency ratios between the tempered notes of s and the tonic. The existence of a solution with relatively simple ratios and relatively simple minimal generators indicates that scale s is a possible candidate for musical use.

6.1   Scales on a 12-Note Chromatic

We began by analyzing scales on the classical 12 chromatic tones, since they can be performed on traditional instruments. The results for 7-note scales appear in Table 2, which shows the number of solutions (a, b) found for each of the 14 possible scales.[4] Since there are multiple solutions for each scale, the table displays a solution in which the ratios are simplest (sometimes there are 2 or 3 solutions in which the ratios are more or less equally simple). It also shows the minimal generators for each scale. Most of these scales correspond to the classical Greek modes and/or modern major and minor scales, as indicated in the table. Interestingly, the classical modes are precisely the scales that can be obtained from the single generator $3/2$.[5]

We also investigated nonclassical scales with 6, 8 or 9 notes (Table 3). The only 6-note scale is the whole-tone scale, whose musical possibilities are limited. There are only two 8-note scales, each of which has three keys. The first of the two might be viewed as superior, because it contains both the major third and the fifth, neither of which occurs in the second. However, the second has a half-step leading tone to the tonic (i.e., $s_8 = 0$), which may be viewed as desirable because it allows for stronger cadences. There are thirty 9-note scales, and these tend to contain a large number of consonant ratios, giving them a distinctive sound. Table 3 displays the 10 scales that begin with a whole tone. Scales 22 and 23 seem especially appealing for composition due to their distribution of semitones and simple ratios. They are identical, except that one has a major sixth and one a dominant seventh interveral. The author wrote an extended work for organ using scale 23 [8].

---

[4] We follow the convention of numering the scales in the order of the tuples s treated as binary numbers.

[5] For the Dorian, a solution with generators $3/2$ and $5/3$ is shown because it results in simpler ratios. The single generator $3/2$ results in ratios $9/8$, $32/27$, $4/3$, $3/2$, $27/16$, $16/9$.

Table 2. The fourteen 7-note scales on a 12-note chromatic. The number of solutions obtained is shown for each scale, followed by one selected solution with relatively simple ratios and minimal generators. The solutions are generated with a maximum denominator of $M = 32$. All scales have 12 keys.

	Scale	Solns	Ratios with tonic	Minimal generators	
1	0101111	27	$\frac{1}{1}\ \frac{16}{15}\ \frac{6}{5}\ \frac{5}{4}\ \frac{45}{32}\ \frac{8}{5}\ \frac{16}{9}$	$\frac{5}{3}\ \frac{3}{2}\ \frac{3}{2}\ \frac{5}{4}\ \frac{9}{8}\ \frac{3}{2}\ \frac{5}{3}$	
2	0110111	10	$\frac{1}{1}\ \frac{18}{17}\ \frac{6}{5}\ \frac{4}{3}\ \frac{24}{17}\ \frac{8}{5}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}$	Locrian mode
3	0111011	18	$\frac{1}{1}\ \frac{16}{15}\ \frac{6}{5}\ \frac{4}{3}\ \frac{3}{2}\ \frac{8}{5}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}$	Phrygian mode
4	0111101	26	$\frac{1}{1}\ \frac{16}{15}\ \frac{6}{5}\ \frac{4}{3}\ \frac{3}{2}\ \frac{5}{3}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{5}{3}\ \frac{3}{3}\ \frac{3}{2}\ \frac{3}{2}\ \frac{5}{3}\ \frac{3}{2}$	
5	1010111	6	$\frac{1}{1}\ \frac{9}{8}\ \frac{6}{5}\ \frac{4}{3}\ \frac{45}{32}\ \frac{8}{5}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{5}{2}\ \frac{3}{4}\ \frac{5}{2}\ \frac{3}{4}\ \frac{3}{2}\ \frac{3}{2}$	
6	1011011	6	$\frac{1}{1}\ \frac{9}{8}\ \frac{6}{5}\ \frac{4}{3}\ \frac{3}{2}\ \frac{8}{5}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}$	Aeolian mode (natural minor)
7	1011101	10	$\frac{1}{1}\ \frac{9}{8}\ \frac{6}{5}\ \frac{4}{3}\ \frac{3}{2}\ \frac{5}{3}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{3}{2}\ \frac{3}{2}$	Dorian mode
8	1011110	27	$\frac{1}{1}\ \frac{9}{8}\ \frac{6}{5}\ \frac{4}{3}\ \frac{3}{2}\ \frac{5}{3}\ \frac{15}{8}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{5}{2}$	melodic minor
9	1101011	14	$\frac{1}{1}\ \frac{9}{8}\ \frac{5}{4}\ \frac{4}{3}\ \frac{3}{2}\ \frac{8}{5}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{9}{8}$	
10	1101101	9	$\frac{1}{1}\ \frac{9}{8}\ \frac{5}{4}\ \frac{4}{3}\ \frac{3}{2}\ \frac{5}{3}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}$	Mixolydian mode
11	1101110	17	$\frac{1}{1}\ \frac{9}{8}\ \frac{5}{4}\ \frac{4}{3}\ \frac{3}{2}\ \frac{5}{3}\ \frac{15}{8}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}$	Ionian mode (major)
12	1110101	10	$\frac{1}{1}\ \frac{9}{8}\ \frac{5}{4}\ \frac{45}{32}\ \frac{3}{2}\ \frac{5}{3}\ \frac{16}{9}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}$	
13	1110110	16	$\frac{1}{1}\ \frac{9}{8}\ \frac{5}{4}\ \frac{45}{32}\ \frac{3}{2}\ \frac{5}{3}\ \frac{15}{8}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}$	Lydian mode
14	1111010	34	$\frac{1}{1}\ \frac{9}{8}\ \frac{5}{4}\ \frac{45}{32}\ \frac{5}{3}\ \frac{15}{8}$	$\frac{5}{3}\ \frac{5}{3}\ \frac{3}{2}\ \frac{5}{2}\ \frac{3}{4}\ \frac{3}{2}\ \frac{2}{2}$	

Table 3. The 6-note whole-tone scale, two 8-note scales, and 10 of the 30 9-note scales on a 12-note chromatic. Solutions are generated with maximum denominator $M = 32$.

Scale	Solns	Keys	Ratios with tonic									Minimal generators								
1.111111	6	2	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{5}{4}$	$\frac{45}{32}$	$\frac{8}{5}$	$\frac{16}{9}$				$\frac{5}{4}$	$\frac{5}{4}$	$\frac{5}{4}$	$\frac{5}{4}$	$\frac{5}{4}$	$\frac{9}{5}$		
1.01010101	>50	3	$\frac{1}{1}$	$\frac{16}{15}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{45}{32}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{16}{9}$		$\frac{3}{2}$	$\frac{5}{3}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{9}{8}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$
2.10101010	>50	3	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{4}{3}$	$\frac{45}{32}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{15}{8}$		$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{5}{3}$	$\frac{3}{2}$
21.100001010	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{45}{32}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$ $\frac{3}{2}$
22.100010010	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$ $\frac{3}{2}$
23.100010100	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{16}{9}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$ $\frac{3}{2}$
24.100100010	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{45}{32}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$ $\frac{3}{2}$
25.100100100	>50	4	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{45}{32}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{16}{9}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{9}{5}$ $\frac{3}{2}$
26.100101000	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{45}{32}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{16}{9}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{9}{8}$ $\frac{3}{2}$
27.101001010	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{4}{3}$	$\frac{45}{32}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$ $\frac{3}{2}$
28.101000100	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{45}{32}$	$\frac{8}{5}$	$\frac{16}{9}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$ $\frac{3}{2}$
29.101001000	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{4}{3}$	$\frac{45}{32}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{16}{9}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{5}{3}$ $\frac{3}{2}$
30.101010000	>50	12	$\frac{1}{1}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{4}{3}$	$\frac{45}{32}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{16}{9}$	$\frac{15}{8}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$ $\frac{3}{2}$

6.2   Scales on a 19-Note Chromatic

We now consider nonclassical chromatic scales. An obvious question is how many chromatic notes result in attractive scales. One initial screening is to investigate which chromatics contain tones with several simple ratios with the tonic (within tolerance), because these ratios then become available for the scales. Table 4 shows the simple ratios that occur in various chromatic scales. The 19-tone scale stands out as clearly superior. It is the the only scale that strictly dominates the classical 12-tone scale, containing its simple ratios plus three more. The 24-note scale (quarter tones) obviously contains all the simple ratios of the classical scale, but no more, and so there is no compelling reason to move to quarter tones. We therefore concentrate on the 19-note chromatic scale.

Table 4. Simple ratios (indicated by heavy black dots) that occur in chromatic scales having 6 to 24 notes.

Ratio	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3/2	·	·	·	·	·	·	•	·	·	·	·	•	·	•	·	·	•	·	•
4/3	·	·	·	·	·	·	•	·	·	·	·	•	·	•	·	·	•	·	•
5/3	·	·	•	·	·	•	•	·	·	•	•	·	·	•	•	·	•	•	•
5/4	•	·	·	•	·	·	•	·	·	•	•	·	•	•	·	•	·	·	•
7/4	·	·	·	·	•	•	·	·	·	•	•	·	·	•	·	•	·	·	·
6/5	·	·	·	·	·	•	·	·	·	•	·	·	·	•	·	·	•	•	·
7/5	·	·	·	·	·	·	·	·	·	·	·	·	·	•	·	•	·	•	·
8/5	•	·	·	•	·	·	•	•	·	•	•	·	•	•	·	•	•	·	•
9/5	·	•	·	·	·	·	·	•	•	·	·	·	·	•	•	•	·	·	·

In particular, we study 11-note diatonic scales, which contain 3 semitones, or one more semitone than the classical 7-note scales. Since the 19 chromatic tones are already rather closely spaced, it seems desirable to limit the number of semitones in the scale. We therefore rule out 12-note scales, which contain 5 semitones, meaning that almost half of the intervals are semitones. On the other hand, 10-note scales have only one semitone and are therefore nearly whole-tone scales and of limited musical interest. Eleven-note scales seem a good compromise.

There are 77 11-note diatonic scales on a 19-note chromatic that satisfy our criteria. Since we selected the 19-note chromatic due to its inclusion of many simple ratios, it is reasonable to concentrate on scales in which most of these simple ratios occur. Table 5 displays, for each of the 77 scales, the largest subset of simple ratios that occur in at least one solution. Thirty-seven different subsets of ratios appear in the scales, corresponding to the columns of Table 5. They partition the scales into 37 equivalence classes, labeled A–Z and a–k.

As it happens, all of the classes are dominated by the four classes A, E, P and W (indicated by boldface in Table 5), in the sense that the simple ratios that occur in any class also occur in one of these four. We therefore concentrate on these classes, which collectively contain 9 scales. Table 3 displays two solutions for a selected scale in each of the four classes. The first solution shown is one with the smallest minimal generators (i.e., one for which the largest minimal generator is smallest). The second solution is one with the simplest ratios with the tonic. The two solutions are very similar but indicate alternative ways the ear can interpret the more complicated pitch ratios.[6]

Scale 72 (class A) contains the most simple ratios with the tonic, including a fifth, fourth, major third, major sixth, minor sixth, and two additional intervals with ratios 7/5 and 9/5. Scale 7 (class E) lacks the fourth and the 9/5 ratio, but

---

[6] A complete list of all 50 solutions found for each of the 77 scales is available at web.tepper.cmu.edu/jnh/music/scales11notes19.pdf and as electronic supplementary material published online with this article.

Table 5. Occurrences of simple ratios in 11-note scales on a 19-note chromatic. Each column corresponds to a class of scales as indicated in the key at the bottom.

Ratio	**A**	B	C	D	**E**	F	G	H	I	J	K	L	M	N	O	**P**	Q	R	S	T	U	V	**W**	X	Y	Z	a	b	c	d	e	f	g	h	i	j	k
3/2	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
4/3	•	•	•	•	•	•	·	•	•	•	•	·	·	·	·	•	·	·	·	·	·	·	•	•	•	•	•	•	•	•	•	·	·	·	·	·	·
5/3	•	•	•	•	·	•	·	·	·	·	·	·	·	·	·	•	·	•	•	·	·	·	•	·	·	·	·	·	·	·	•	•	•	·	·	·	·
5/4	•	•	•	•	•	•	•	•	•	·	·	•	·	·	·	•	·	·	·	•	·	·	•	·	·	·	•	•	•	·	·	•	•	·	·	•	·
7/4	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
6/5	·	·	·	·	•	•	·	·	•	•	·	·	•	•	·	•	·	•	•	·	·	•	·	·	·	·	•	·	·	·	·	·	•	•	·	·	•
7/5	•	•	•	·	•	·	•	•	·	·	·	•	•	·	·	•	•	·	•	·	•	•	·	·	·	·	·	·	·	·	•	•	•	•	•	•	•
8/5	•	·	·	•	•	•	•	•	•	•	·	•	·	·	•	·	·	·	•	•	•	·	·	·	·	·	·	•	•	·	•	·	·	·	·	•	•
9/5	•	•	·	•	·	·	·	•	·	•	·	•	•	·	•	•	·	•	·	•	·	•	·	·	·	•	·	•	·	·	•	•	·	•	·	·	·

A - 72	K - 12,43	U - 57	e - 13,29,44
B - 69,70,71	L - 28	V - 42	f - 60,61
C - 68	M - 65,66	**W** - 26,27	g - 59
D - 74,75	N - 63,64	X - 10,11,25	h - 18,35,36,50,51,54
E - 7,8	O - 62	Y - 5,6	i - 17,34,49
F - 22,23	**P** - 40,41,55,56	Z - 15,31,32,46,47	j - 58
G - 73	Q - 20,21,38,39,53	a - 14,30,45	k - 16,33,48
H - 2	R - 19,37,52	b - 9,24	
I - 76	S - 67	c - 77	
J - 3	T - 1	d - 4	

it contains a minor third. It also lacks a half-step leading tone. Scale 56 (class P) contains as many simple ratios as scale 7, but it lacks the major third, which might be regarded as a weakness. Scale 27 (class W) lacks the fifth, perhaps a greater weakness.

The key structure of the 11-note scales contrasts significantly with that of the classical scales, as indicated in Table 7. The table shows the distance of each key from the tonic, where distance is measured by m minus the number of notes in common (recall that m is the number of notes in the scale). In the classical case, the two most closely related keys start on the two most consonant intervals, the fourth and fifth. In scale 72, the closest key starts on the major sixth, while the keys starting on the fourth and fifth are among the most distant. In scale 7,

Table 6. Four of the 77 11-note scales on a 19-note chromatic. Each scale has 19 keys. At least 50 solutions were obtained for each scale, of which 2 are shown. The solutions were generated with maximum denominator $M = 64$.

Scale	Class	Ratios with tonic												Minimal generators									
7.01101011111	E	$\frac{1}{1}$	$\frac{25}{24}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{7}{5}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{12}{7}$	$\frac{25}{18}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{7}{4}$	$\frac{3}{2}$
		$\frac{1}{1}$	$\frac{36}{35}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{7}{5}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{12}{7}$	$\frac{13}{17}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{2}$	$\frac{3}{2}$	$\frac{4}{2}$	$\frac{7}{4}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{13}{7}$
27.10101111110	W	$\frac{1}{1}$	$\frac{15}{14}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{10}{7}$	$\frac{54}{35}$	$\frac{5}{3}$	$\frac{9}{5}$	$\frac{27}{14}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{4}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{4}{2}$	$\frac{3}{2}$	$\frac{5}{4}$
		$\frac{1}{1}$	$\frac{16}{15}$	$\frac{9}{8}$	$\frac{6}{5}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{10}{7}$	$\frac{14}{9}$	$\frac{5}{3}$	$\frac{9}{5}$	$\frac{35}{18}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{4}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{7}{4}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{5}{4}$
56.11011110110	P	$\frac{1}{1}$	$\frac{15}{14}$	$\frac{7}{6}$	$\frac{6}{5}$	$\frac{9}{7}$	$\frac{7}{5}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{9}{5}$	$\frac{27}{14}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$
		$\frac{1}{1}$	$\frac{13}{12}$	$\frac{7}{6}$	$\frac{6}{5}$	$\frac{9}{7}$	$\frac{7}{5}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{9}{5}$	$\frac{35}{18}$	$\frac{3}{2}$	$\frac{13}{7}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{7}{5}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$
72.11110110110	A	$\frac{1}{1}$	$\frac{16}{15}$	$\frac{7}{6}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{7}{5}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{9}{5}$	$\frac{35}{18}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{5}{3}$
		$\frac{1}{1}$	$\frac{15}{14}$	$\frac{7}{6}$	$\frac{5}{4}$	$\frac{4}{3}$	$\frac{7}{5}$	$\frac{3}{2}$	$\frac{8}{5}$	$\frac{5}{3}$	$\frac{9}{5}$	$\frac{27}{14}$	$\frac{3}{2}$	$\frac{7}{5}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{5}{3}$	$\frac{9}{5}$

the closest key starts a step below the tonic, while in scales 27 and 56, it starts a step above the tonic. In scale 72, the second closest key starts a step above the tonic. This means that in all of these 11-note scales, one can wander further from the tonic (up to a point) by taking steps up or down, as opposed to following the cycle of fourths or fifths as in the traditional scales. The table also shows scale 23 discussed earlier, whose key structure is again very different. All of the alternate keys have a distance 2 or 3 from the tonic key. Some other 11-note scales have keys with a distance 1 from the tonic key; namely, scales 9, 13, 14, 30, 34, 35, 50, 53, 54, 64, and 66.

Table 7. Key structure of selected scales, showing distance of each key from the tonic. The interval m3rd is a minor third.

Classical major scale

Note	1	1♯	2	2♯	3	4	4♯	5	5♯	6	6♯	7
Interval			2nd		3rd	4th		5th		6th		7th
Distance	0	5	2	3	4	1	5	1	4	3	2	5

Scale 23 of 9 notes on 12-note chromatic

Note	1	1♯	2	3	4	5	6	7	7♯	8	9	
Interval			2nd	m3rd	3rd	4th	5th	m6th		m7th	7th	
Distance	0	3	3	2	2	2	3	2	2	2	3	3

Scale 7 of 11 notes on 19-note chromatic

Note	1	2	2♯	3	3♯	4	5	5♯	6	7	7♯	8	8♯	9	9♯	10	10♯	11	11♯
Interval				2nd		m3rd	3rd		4th			5th		m6th					
Distance	0	8	3	5	5	4	5	5	4	5	5	4	5	5	4	5	5	3	8

Scale 27 of 11 notes on 19-note chromatic

Note	1	1♯	2	3	3♯	4	5	5♯	6	6♯	7	7♯	8	8♯	9	9♯	10	10♯	11
Interval			2nd		m3rd	3rd	4th								6th				
Distance	0	8	3	5	4	6	3	6	4	5	5	4	6	3	6	4	5	3	8

Scale 56 of 11 notes on 19-note chromatic

Note	1	1♯	2	2♯	3	4	4♯	5	5♯	6	6♯	7	7♯	8	9	9♯	10	10♯	11
Interval						m3rd						5th	m6th	6th					
Distance	0	8	3	5	6	2	7	3	6	4	4	6	3	7	2	6	5	3	8

Scale 72 of 11 notes on 19-note chromatic

Note	1	1♯	2	2♯	3	3♯	4	4♯	5	6	6♯	7	7♯	8	9	9♯	10	10♯	11
Interval								3rd	4th			5th	m6th	6th					
Distance	0	8	3	5	6	2	7	3	6	4	4	6	3	7	2	6	5	3	8

We can also contrast the harmonic structure of the 11-note scales with that of the classical major scale. The basic triads in the classical scale are the major triad, with ratios 4:5:6, and the minor triad 10:12:15. The primary quadrads are the major seven chord 8:10:12:15, the minor seven 10:12:15:18, and the all-important dominant seven 36:45:54:64. The rather dissonant dominant seven chord is not so much inspired by harmonic considerations as by the ubiquitous passing tone from the fifth to the third in cadences, which creates a seven chord with the dominant triad.

The 11-note scales differ harmonically in two major respects: the disappearance of the dominant seven, and the addition of exotic harmonies with simple ratios. For definiteness, we focus on scale 72, which contains the largest collection

of simple ratios. While the dominant seven chord 36:45:54:64 occurs in some nonstandard scales (such as 9-note scales 23, 25 and 26 in Table 3), it does not occur in scale 72. This suggests that cadences could look very different than in classical scales.

Like the classical major scale, scale 72 contains the major and minor triads (notes 1-4-7 and 5-8-12) as well as the minor seven chord (9-12-15-18), although it lacks the major seven chord. It presents several new harmonies with simple ratios as well. There are three triads that might be viewed as compressed minor triads, and that extend nicely to quadrads. One has ratios 5:6:7 that extend to 5:6:7:9 (notes 9-12-14-18), a second has ratios 6:7:8 that extend to 6:7:8:10 (notes 1-3-5-9), and a third has ratios 7:8:10 that extend to 7:8:10:12 (notes 3-5-9-12). The scale also has a quadrad that is similar to a dominant seven chord (notes 5-9-12-15), except that it has a flatter seventh and much simpler ratios 4:5:6:7.

A final question is whether the "tensions" that are widely used in jazz harmony have a parallel in 11-note scales. Tensions are usually formed by adding notes that are a major ninth above notes of an existing chord [3]. As an example, a major seven chord 1-3-5-7 is extended to 1-3-5-7-9-11♯-13. There does in fact seem to be a parallel to tensions in scale 72, except that they are formed by adding notes whose ratio to the next lower note is 6/5. In this way, we can extend the major triad 1-4-7 to 1-4-7-13-15-18-21, with all notes within the same key (the next note 24♭ moves to a different key). The ratios are exact, except that we must slightly adjust the tension ratio 54/25 of note 13 to 32/15, which is the ratio for this note implied by one of the two solutions of Table 6.

7   Conclusion

We developed a method for systematically generating alternative diatonic scales that share two important characteristics of classical 7-note scales: intervals that correspond to simple ratios, and multiple keys based on a tempered chromatic scale. We defined these characteristics mathematically, and in particular we recursively defined suitable pitch ratios as those that can be obtained from other ratios using a small set of "generators." This approach was partially vindicated by the fact that within the classical 12-note chromatic, the scales that can be obtained entirely from the simplest generator (3/2) are precisely the classical Greek modes, which include the modern major and natural minor scales.

We found our criteria to be well suited for formulation in a constraint satisfaction model and therefore used a constraint programming solver to search for acceptable scales. We considered tempered chromatic scales having from 6 to 24 tones and observed that two of them stand out as superior with respect to the number of simple ratios they contain: the classical 12-note chromatic and the 19-note chromatic. This allowed us to narrow the range of search by concentrating on scales based on these two chromatics.

We first studied scales on the 12-tone chromatic having 6, 7, 8, and 9 notes and identified two 9-note scales that, aside from the classical modes, seem particularly appealing. We focused most of our effort, however, on exploring

scales on the 19-note chromatic. Scales with 11 notes appear to be the most promising, and 9 of these 77 scales most deserve attention due to the number of simple ratios they contain. We found these scales to provide significant musical resources that are not available in classical scales, including a contrasting and more complex key structure, as well as a number of new harmonies.

In particular, the most attractive 11-note scale contains, in addition to the classical major and minor triads, three triads and four quadrads with simple ratios that do not appear in traditional scales. These provide many new possiblities for harmonic texture. The scale contains no dominant seven chord, which suggests that it would inspire very different chord progressions than the classical major and minor scales. In addition, it supports complex tensions that are analogous to but harmonically different from those commonly occurring in jazz arrangements.

We conclude that this and the other 11-note scales we singled out could take music in an interesting new direction, and we suggest them to composers as possibly worthy of experimentation. Such experiments would presumably rely primarily on electronic synthesizers, due to the difficulty of building acoustic instruments that support nonstandard scales. It is essential, however, not to generate tones as sine waves or other simplified wave forms. The tones should carry a full complement of upper harmonics that mimic those that would be generated by acoustic instruments, because otherwise the exotic intervals and harmonies of these scales cannot be easily recognized or appreciated.

Appendix

A Chorale and Fugue for organ [8] uses scale 23 on 9 notes. The chorale cycles through the tonic (A) and the two most closely related keys (C♯, F). The cadences illustrate that dominant seven chords need not play a role, even though they occur in the scale. Rather, the cadences use two leading tones and pivot on the tonic, often by moving from the lowered submediant. The chorale is followed by a double fugue that again cycles through the three keys A, C♯, F. The first subject enters on these pitches but without a key change. The second subject (bar 96) illustrates the expanded possibilities for suspensions and pivots.

References

1. Balzano, G.J.: The graph-theoretic description of 12-fold and microtonal pitch systems. Comput. Music J. **4**, 66–84 (1980)
2. Benson, D.J.: Music: A Mathematical Offering. Cambridge University Press, Cambridge (2006)
3. Berkman, D.: The Jazz Harmony Book. Sher Music Company, Petaluma (2013)
4. von Bohlen, H.: Tonstufen in der Duodezine. Acustica **39**, 76–86 (1978)
5. Brown, M.: Explaining Tonality: Schenkerian Theory and Beyond. University of Rochester Press, Rochester (2005)
6. Chew, E.: Mathematical and Computational Modeling of Tonality: Theory and Application. Springer, New York (2014)

7. Gould, M.: Balzano and Zweifel: Another look at generalized diatonic scales. Perspect. New Music **38**, 88–105 (2000)
8. Hooker, J.N.: Chorale and fugue on a 9-note scale, for organ (2013). web.tepper.cmu.edu/jnh/music, with mp3 file
9. Mathews, M.V., Pierce, J.R., Reeves, A., Roberts, L.A.: Theoretical and experimental exploration of the Bohlen-Pierce scale. J. Acoust. Soc. Am. **68**, 1214–1222 (1988)
10. Messiaen, O.: The Technique of My Musical Language. Alphonse Leduc, Paris (1944)
11. Noll, T.: The topos of triads. In: Fripertinger, H., Reich, L. (eds.) Colloquium on Mathematical Music Theory, pp. 1–26 (2005)
12. Noll, T.: Musical intervals and special linear transformations. J. Math. Music **1**, 121–137 (2007)
13. Noll, T.: Getting involved with mathematical music theory. J. Math. Music **8**, 167–182 (2014)
14. Oswald, J.: Introduction to the Theory of Heinrich Schenker: The Nature of the Musical Work of Art. Longman, New York (1982)
15. Pearce, M.: The group-theoretic description of musical pitch systems, City University, London (2002)
16. Perret, W.: Some Questions of Musical Theory. W. Heffer & Sons, Cambridge (1926)
17. Pierce, J.R.: Attaining consonance in arbitrary scales. J. Acoust. Soc. Am. **40**, 249 (1966)
18. Pierce, J.R.: Consonance and scales. In: Cook, P.R. (ed.) Music, Cognition, and Computerized Sound: An Introduction to Psychoacoustics, pp. 168–184. MIT Press (2001)
19. Plomp, R., Levelt, W.J.M.: Tonal consonance and critical bandwidth. J. Acoust. Soc. Am. **38**, 548–560 (1965)
20. Sethares, W.A.: Local consonance and the relationship between timbre and scale. J. Acoust. Soc. Am. **94**, 1218–1228 (1993)
21. Sethares, W.A.: Tuning, Timbre, Spectrum, Scale. Springer, London (2005)
22. Zweifel, P.F.: Generalized diatonic and pentatonic scales: a group-theoretic approach. Perspect. New Music **34**, 140–161 (1996)

Assisted Lead Sheet Composition
Using FlowComposer

Alexandre Papadopoulos[1,2]([✉]), Pierre Roy[1], and François Pachet[1,2]

[1] Sony CSL, 6 rue Amyot, 75005 Paris, France
`roy@csl.sony.fr, pachetcsl@gmail.com`
[2] Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6,
75005 Paris, France
`alexandre.papadopoulos@lip6.fr`

Abstract. We present FlowComposer, a web application that helps users compose musical lead sheets, i.e. melodies with chord labels. Flow-Composer integrates a constraint-based lead sheet generation tool in which the user retains full control over the generation process. Users specify the style of the lead sheet by selecting a corpus of existing lead sheets. The system then produces a complete lead sheet in that style, either from scratch, or from a partial lead sheet entered by the user. The generation algorithm is based on a graphical model that combines two Markov chains enriched by Regular constraints, representing the melody and its related chord sequence. The model is sampled using our recent result in efficient sampling of the Regular constraint. The paper reports on the design and deployment of FlowComposer as a web-service, part of an ecosystem of online tools for the creation of lead sheets. FlowComposer is currently used in professional musical productions, from which we collect and show a number of representative examples.

Keywords: Music generation · Graphical models · Belief propagation · Sampling · Web-service · User interaction

1 Introduction

Modelling polyphonic music is a particularly challenging task in artificial intelligence. This is probably because music, even in its simplest form, manifests itself under many interdependent dimensions, such as melody (successions of notes in time), harmony (simultaneous notes or chord labels) and meter (constraints on durations of notes making up a bar for instance). Constraint programming has been extensively studied to model polyphonic music [1]. However CSP approaches require expert musicians to encode explicitly the rules (e.g. of harmony) as constraints, and this task is not always possible nor desirable, as these constraints usually correspond to a fixed and slightly outdated musical style.

Recent advances in machine-learning and graphical models have managed to model all these dimensions in single statistical models, such as deep networks [3]. These models have been shown to be able to capture various statistical properties

of musical style. However, they are difficult to control, and have not developed into systems mature enough to be used outside specific demos.

In this paper, we address a specific case of polyphonic music: lead sheets. Lead sheets consist in monophonic melodies, augmented with chord labels (see Fig. 5). Lead sheets are routinely used in popular music, including pop, rock, jazz or Brazilian music. Lead sheets also have a strong commercial value as they are the primary asset of music publishing companies. We describe an application called FlowComposer, a lead sheet composition tool. The basic concept is to provide an online lead sheet editor enriched with style imitation generation capabilities. With FlowComposer, users can generate fully-fledged lead sheets based on partially specified information, that conform to the style of a given composer (or set of lead sheets). Generated lead sheets satisfy several types of constraints: (1) user constraints, (2) metrical constraints, and (3) style constraints. Technically, this paper has three contributions. First, we show how to sample metrically constrained Markov sequences, using our recently introduced model [12]. Then, we show how to exploit this framework to enforce stochastic temporal constraints. Finally, we define a two-voice model for chord and note generation: each voice is a metrically constrained Markov sequence, and we synchronise the two voices using stochastic temporal constraints, to enforce harmony. FlowComposer is a working application that is being used in professional music projects (see Sect. 4.3). The paper describes the technical challenges addressed in designing and deploying FlowComposer and some interesting uses of the system.

2   Background on Constrained Markov Models

Markov models have long been used to generate music in the style of a composer [4,8]. A Markov model can be estimated from a musical corpus, by counting transitions between successive elements. A random walk in a Markov model produces new sequences according to those transition probabilities, but does not, in general satisfy any other desirable property, such as unary constraints (specific values imposed at specific indexes of the sequences) or meter.

We have shown that the formulation of Markov processes as constraints opens the door to fine-grained control over generated sequences [11]. Additional properties such as unary constraints, meter and many others can then be enforced in polynomial time [9,15].

2.1   Enforcing Meter on Markov Sequences

METER is a global constraint that enforces a *metrical structure* on a sequence of temporal events [15]. Let $X_1, \ldots, X_n$ be a sequence of temporal events, such as notes, words, tasks, etc. Let $d(X_i)$ be the integer duration of the element assigned to X_i. We define $o(X_i)$, the *onset* or starting time of X_i in the sequence. It is equal to 0 if $i = 1$, or $\Sigma_{j=1}^{i-1} d(X_j)$ for higher indexes. For example, consider a sequence X_1, X_2, X_3, X_4 of integers $[1, 1, 2, 2]$, where the duration of an element

is its own value. The onset of X_1 is 0, the onset of X_2 is 1, the onset of X_3 is $d(X_1) + d(X_2) = 3$, and so on.

Meter Constraint: A METER constraint takes as parameters a *total duration D*, and a *predicate* $\pi(o, e)$, where o is an integer onset, and e an element from the domain of the X_i variables. METER holds on $X_1, \ldots, X_n$ if $\Sigma_{i=1}^{n} d(X_i) = D$, i.e. the sequence has a total duration of D, and $\pi(o(X_i), X_i)$ holds for every element X_i of the sequence, i.e. it is acceptable to start element X_i at time $o(X_i)$, for all elements of the sequence.

For example, suppose we want to create sequences of integers in $\{1, 2, 4\}$ (again, their duration is their own value), summing to 8, and sectioned into groups summing to 4. A solution is $[1, 2, 1, 2, 2]$: the total duration is 8, and it can be sectioned into $[1, 2, 1]$ and $[2, 2]$, each lasting 4. Another solution is $[4, 4]$. Conversely, $[2, 4, 2]$ is not a solution because it cannot be sectioned into subsequences summing to 4. We can encode this problem with METER, by setting $D = 8$, and defining $\pi(o, e)$ as follows:

$$\pi(o, e) \equiv \left(\left\lfloor \frac{o}{4} \right\rfloor = \left\lfloor \frac{o+e}{4} \right\rfloor \right) \vee \left(o + e = 4 \cdot \left(\left\lfloor \frac{o}{4} \right\rfloor + 1 \right) \right)$$

Intuitively, the predicate accepts value e at onset o only if e starts and ends in the same section (first case of the disjunction), or if e ends exactly at the start of the next section (second case). In general, sequences summing to 8 can involve a varying number of elements. In order to encode such sequences with a fixed number of variables, we choose a length sufficient for the longest sequence of total duration D, and we introduce a dummy element of zero duration, hereafter called *padding element*, used to fill the remainder of a sequence when the target duration is reached with fewer elements. In our example, we need at most 8 variables, to represent the sequence $[1, 1, 1, 1, 1, 1, 1, 1]$. The previous solutions are encoded as $[1, 2, 1, 2, 2, 0, 0, 0]$ and $[4, 4, 0, 0, 0, 0, 0, 0]$, respectively. In order to restrict the padding element to the end of the sequence, we need the predicate to also satisfy the condition $(o = D) \Rightarrow (e = 0)$.

We illustrate METER with "Frère Jacques", a French nursery rhyme, also known in English as "Brother John". We show the first 4 bars of this melody on Fig. 1. This melody satisfies the following metrical constraints: its total duration, determined by the number of bars and the time signature, is equal to 16 beats (4 bars of 4 beats each), and notes do not cross bar lines.

In [15], we showed how to propagate METER as a global constraint in a CSP. However, an important observation underlying this work is that METER can also be formulated as a REGULAR constraint. As a result, we can apply the technique

Fig. 1. The first 4 bars of the Frère Jacques melody

described in the next section to correctly sample metrically constrained Markov sequences, a novel result in this paper.

2.2   Markov Models and Regular Constraints

Recently, we generalised Markov constraints to REGULAR constraints [13], specifying that a Markov sequence $X_1, \ldots, X_n$ should additionally form a word from a *regular language* $\mathcal{L}(\mathcal{A})$, recognised by an imposed *finite-state automaton* $\mathcal{A}$. We use a factor-graph based model to encode this constrained Markov model, and use *belief propagation* to sample, with unbiased probabilities, Markov sequences satisfying REGULAR, in polynomial time [12]. Belief propagation generalises constraint propagation, where instead of propagating information on value consistency, we propagate probabilities associated with values.

A Markov model is a stochastic process, where the probability for state X_i, a random variable, depends only on the last state X_{i-1}. Each random variable X_i takes values amongst an *alphabet*, denoted $\mathcal{X}$. A Markov model produces sequence $X_1, \ldots, X_n$ with probability $P(X_1) \times P(X_2|X_1) \times \cdots \times P(X_n|X_{n-1})$. Given additional unary constraints $P_i(X_i)$ and a REGULAR constraint specified by an automaton $\mathcal{A}$, the problem of sampling a Markov sequence subject to REGULAR is defined as the problem of sampling from the following distribution:

$$p_{target}(X_1, \ldots, X_n) \propto \begin{cases} \prod_{i=2}^{n} P(X_i|X_{i-1}) \times \prod_{i=1}^{n} P_i(X_i) & \text{if } X_1 \cdots X_n \in \mathcal{L}(\mathcal{A}) \\ 0 & \text{otherwise} \end{cases}$$

The symbol $\propto$ ("proportional to") indicates that the equality holds after normalisation. The first case indicates that the regular constraint holds (i.e. the sequence belongs to the specified language), and in the expression, the *unary factors* $P_i(X_i)$ are distributions that generalise unary constraints on the variables X_i. A unary factor merely biases the probability of the overall sequence, but does not necessarily correspond to the marginal distribution on X_i of the resulting distribution. In order to sample p_{target}, we reformulate it into a distribution p_{reg} of $Y_1, \ldots, Y_n$, where the new Y_i variables take values (e, q), where $e \in \mathcal{X}$ is a state of the Markov chain, and q is a state of the automaton $\mathcal{A}$ that defines the REGULAR constraint. Sampling p_{target} is equivalent to sampling p_{reg}, and projecting each resulting sequence $(e_1, q_1), \ldots, (e_n, q_n)$ to $e_1, \ldots, e_n$. We show in [12] that p_{reg} can be represented as a tree-structured factor-graph, and therefore that we can use belief propagation to sample p_{reg} in polynomial time. The time complexity of this procedure is in the size of the alphabet of Y_i times the length of the sequence, i.e. $O(|\mathcal{X}| \cdot |Q| \cdot n)$.

2.3   Sampling Metrically Constrained Markov Sequences

In order to sample metrically constrained Markov sequences, we need to build the METER automaton $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ as follows:

– Q is the set of states: for each possible temporal position i, from 0 to the target duration D, we build a state q_i;

– q_0 is the initial state, corresponding to temporal position 0;
– F is the subset of Q of accepting states: it contains only the state q_D corresponding to the target duration D;
– Σ is the alphabet of the automaton, and contains the values e in the domains of variables X_i (i.e. musical events);
– δ is the transition function, mapping a state from Q and a symbol from Σ to a destination state: for a state q_o, corresponding to temporal position o, and an element e, we reach state q_d (i.e. $\delta(q_o, e) = q_d$), corresponding to temporal position d, iff $d = o + d(e)$ *and* $\pi(o, e)$ holds.

Figure 2 shows the METER automaton recognising all two-bar sequences we can build using the notes from Fig. 1, and subject to the same metrical constraint. Since METER expects integer durations, durations are rescaled to match integer values, i.e. quarter notes have a duration of 1, half notes have a duration of 2, bars have a duration of 4 and the full melody has a duration of 8.
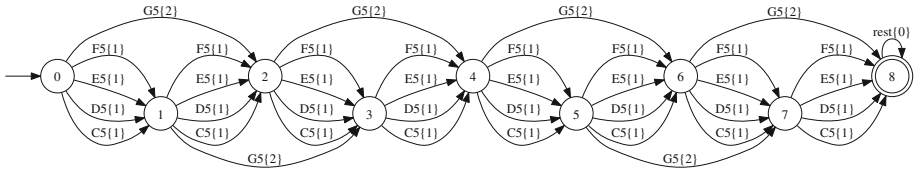
Fig. 2. The automaton accepting 2-bar melodies with the five notes from the melody on Fig. 1, with meter. Discretised durations are shown between curly braces.

The procedure for sampling a metrically constrained Markov sequence is quadratic in the total duration D: the time complexity for sampling a Markov sequence subject to REGULAR is $O(|\mathcal{X}| \cdot |Q| \cdot n)$, as mentioned in Sect. 2.2. We need at most D variables to represent a sequence of total duration D (if the smallest duration is 1), and since each state of the METER automaton corresponds to a temporal position bounded by D, we have $|Q| = D + 1$, and therefore the overall time complexity is $O(|\mathcal{X}| \cdot D^2)$.

3   A Two-Voice Statistical Model of Lead Sheets

FlowComposer is based on a two-voice model of lead sheets, which captures stylistic information concerning the melody, the harmony, and the interaction between harmony and melody (see Fig. 3). The chord model and the melody models are both based on a representation of music meter as a regular constraint as described in the preceding section.

Lead sheets are generated with the following procedure:

1. Generate a chord sequence by sampling the Markov+Meter model on chords, taking into account imposed sections of melodies as harmonic constraints.
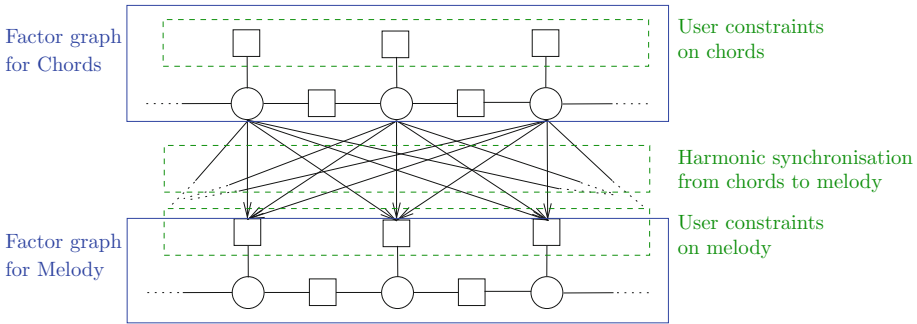
Fig. 3. The two-voice model for lead sheet generation

Additionally, when sampling, instead of drawing chords with their exact marginal distribution, we use a *variable order heuristic* that favours the chords that tend to replicate longer chord sequences from the corpus [2]. In practice, for each context size, we compute the entropy of the distribution on the chords that the context allows, then we choose a context length randomly, with a probability proportional to this entropy. This increases the impression of style imitation for the chord sequence, while avoiding risk of downright plagiarism, since orders with a low entropy are effectively discarded.

2. Generate a melody by sampling the Markov+Meter model on melody, imposing the chord sequence generated in the first step as a *harmonic constraint*. Here we do not use variable order, since each chord typically covers many notes, and the harmonic constraint introduces an amount of higher order correlation between all notes under a certain chord.

3.1 Markov+Meter Model for Chord Sequence

We generate a Markov sequence $X_1, \ldots, X_n$, where X_i is assigned a chord, represented by a chord label and an integer duration. We train an order 1 Markov model on the corpus represented as sequences of chords. We use METER to impose a total duration equal to the duration of the lead sheet to compose, and to forbid chords to cross bar lines.

3.2 Markov+Meter Model for Melody

We generate a Markov sequence $X_1, \ldots, X_n$, where X_i contains a note, represented by its MIDI pitch and integer duration. We obtain integer durations by multiplying all fractional durations with a fixed rescaling factor, equal to the least common multiple of the denominator of all possible fractional durations. This ensures that durations are integer, while maintaining proportions. We train an order 1 Markov model on the corpus represented as sequences of notes. We use METER to impose a total duration equal to that of the lead sheet to compose, multiplied by the rescaling factor, and to forbid notes to cross bar lines.

3.3   Enforcing Harmonic Synchronisation

To generate convincing lead sheets, our model also captures interactions between the note and chord models, in a way that is stylistically consistent with the chosen corpus. Such interactions can be represented in our two-voice model, by exploiting the structure of the factor graphs. We first build a *harmonic model* representing these relations, and then use it to bias sampling.

The Harmonic Model. We define a *harmonic model*, which gives the probability $p_h(n|Ch)$ of placing note n under chord Ch, trained on the corpus. As a consequence, every chord defines a distribution on notes, and this distribution is fully parameterised by the chord label Ch. In order to decrease the amount of data needed to train this model, we adopt a more abstract representation and chords are reduced to their structure alone (for example Fm7 is represented by m7), and we ignore the octave and the duration of a note (for example, A5{2}, the A of the fifth octave, of duration 2, is represented only by A). Technically, for a given observation, we transpose the observed chord to a chord rooted in C with the same structure, and we transpose the observed note accordingly by the same amount of semitones. For example, an observation of a note A5{2} under a chord F m7 is abstracted as E under m7, since there are five semitones between C and F (the chord roots), and equally between E and A (the notes).

Enforcing Stochastic Temporal Constraints. We showed in Sect. 2.3 how to sample METER. We now show that we can define constraints holding on elements specified by their temporal position, rather than by their index in the sequence, a novel result in this paper. We exploit the particular semantics of the METER automaton, i.e. states correspond to temporal positions. Temporal constraints are defined by generalising the METER predicate $\pi(o, e)$ to a *stochastic predicate* $p_\pi(e|o)$. The stochastic predicate defines the probability of placing event e at temporal position o. We then define $\pi(o, e) \equiv (p_\pi(e|o) > 0)$.

We enforce p_π by adding a unary factor in the p_{reg} model, for every Y_i. We recall that the variables Y_i of p_{reg} take values of the form (e, q), with e a value in the alphabet of the Markov chain, and q a state of the automaton. By specifying a unary factor on all variables Y_i, we can bias the probability of an element e appearing with a state q, i.e. at a particular temporal position. The unary factor p_i applied to each Y_i is defined as follows: $p_i(e, q_d) \propto p_\pi(e|o) \cdot p(o)$, where q_d is the state of the Meter automaton corresponding to temporal position $d = o + d(e)$. The probability $p(o)$ gives the probability that o is the start time of an element. We assume it is uniform, but we can also learn this probability from the corpus.

Harmonic Constraints for Melody. Given a chord sequence, we bias the generation of a melody to comply harmonically with the chords. We define the stochastic predicate $p_\pi(n|o) \propto p_h(n|Ch)$, where n is a note, and Ch is the chord occurring at temporal position o.

Harmonic Constraints for Chords. Inferring chord labels from unlabelled melodies has been addressed previously, e.g. using Bayesian inference [14]. In our case, we need to bias the generation of a chord sequence to comply harmonically with the melody. We use a log-likelihood based approach assuming that notes are independent observations and follow the distribution given by p_h. Let us assume that we want to compute the probability $p_\pi(Ch|o)$ of placing chord Ch at temporal position o. Let $n_1, \ldots, n_p$ be the notes of the melody that occur between temporal positions o and $o + d(Ch)$. The average log-likelihood of chord Ch given notes $n_1, \ldots, n_p$ is:

$$\hat{l}(Ch; n_1, \ldots, n_p) = \frac{1}{p} \sum_{i=1}^{p} \log p_h(n_i|Ch)$$

In order to introduce variety in the generated lead sheets, we do not choose the chord with the maximum $\hat{l}$, but rather use this value to define its probability, so that more likely chords are closer to the observed distribution of notes. In practice, we set $p_\pi(Ch|o) \propto \exp\{\hat{l}(Ch; n_1, \ldots, n_p)\}$.

Releasing Harmonic Pressure. The approach we described is often too strict in practice, and, sometimes, we want to relax harmonic pressure. To this end, we propose two strategies. First, we introduce a parameter called *harmonic conformance* that specifies how biased or uniform the harmonic model $p_h(n|Ch)$ should be. The harmonic conformance is a factor α ranging between 0 and 1, and we define a new, relaxed, harmonic model as follow:

$$p'_h(n|Ch) \propto \alpha \cdot p_h(n|Ch) + (1 - \alpha) \cdot p_{uniform}$$

A value of 1 implies strict conformance, a value of 0 results in a uniform distribution, i.e. no harmonic bias at all. This value can be set by the user in the GUI in the form of a slider.

Second, we choose to impose harmony on beats only, as a way of approximating the detection of passing notes, i.e. notes on which harmony is typically less important. In practice, the stochastic predicates $p_\pi(Ch|o)$ and $p_\pi(n|o)$, for chords and notes, are uniform if o is not the start of a beat.

4 Applications

In this section, we describe FlowComposer, a web service for the composition of lead sheets based on the algorithms described in Sect. 3. The web service was implemented using Java. The models for chords, notes and harmony, and the belief propagation procedure to sample those models, have been implemented as an in-house solver. FlowComposer is both an autonomous generator and an interactive music composition application integrated in an ecosystem of online tools for the creation and manipulation of lead sheets. Lead sheets are represented as JSON objects stored in a MongoDB database with more than 12,000

songs in various styles [10]. A graphical lead sheet viewer and editor is implemented as a JavaScript library [7] running in the client web browser. Other services are provided, such as MIDI and audio rendering tools, harmonic and pattern analysis. User sessions and persistence is managed on the server side by a PHP module.

The database covers several genres of popular music: jazz, pop, rock, and Brazilian music, by hundreds of famous composers. A style is defined as a corpus, i.e. a selection of songs from the database, for instance, all the songs by a given composer, in a given genre, or any manual selection. There are 157 types of chords used at least in one song of the database. Among them, 37 chord types have more than 1000 occurrences, e.g., major chords, diminished seventh; 43 occur between 100 and 1000 times, e.g., m69 chords; 61 are used between 20 and 100 times, e.g., m7sus4; and 126 occur fewer than 20 times, e.g., M7♯9.

We show three typical scenarios of the general resolution procedure explained in the preceding section. Section 4.1, describes how FlowComposer can be used as an autonomous lead sheet generator. Section 4.2, describes how FlowComposer can harmonise imposed melodies. Section 4.3, describes how FlowComposer may be used as an online interactive lead sheet composition application.

4.1 Autonomous Generation

The lead sheet generation algorithm may be used to generate lead sheets from scratch, in the style of a given corpus. In this scenario, we parse each song in the training corpus, defining the style of the generated lead sheet. Then, we build and train the Markov+Meter models for the chord and melody generations and the harmonic model. The generation of the lead sheet follows the procedure specified in the preceding section. Note that in the first step, the chord sequence is generated with no harmonic constraints since there is not melody yet. Figure 4 shows an 8-bar lead sheet generated by this procedure in the style of Bill Evans.

Fig. 4. An 8-bar lead sheet generated in the style of Bill Evans

The length of the lead sheet to generate and the training corpus are two input parameters of the generation algorithm. The generation is also influenced by other parameters, such as the number of chord changes, the number of notes, or the harmonic conformance. The 'Number of chord changes' and 'Number of notes' are set to match the average number of notes and chord changes in the corpus. The padding strategy, see Sect. 2.3, allow the system to generate sequences with *approximately* the specified numbers of notes and chord changes. The harmonic conformance α is set by default to its maximum value.

Table 1. Performance of the algorithm for the generation of lead sheet for various lengths and two corpora; ASW stands for American Songwriters, a corpus with 429 songs by composers such as Richard Rodgers, Lorenz Hart, or Irving Berlin

Corpus	Size	Parsing	Length	Training	Next sol.	Model size
ASW	429 songs 630 chords 356 notes	$3''480$	4 bars	$4''$	$1''7$	$731,468$
			8 bars	$9''$	$5''3$	$2,922,436$
			12 bars	$18''$	$12''5$	$6,465,202$
			16 bars	$30''$	$22''$	$11,509,685$
Beatles	45 songs 134 chords 199 notes	$803\,\mathrm{ms}$	4 bars	$600\,\mathrm{ms}$	$600\,\mathrm{ms}$	$245,948$
			8 bars	$1''75$	$1''4$	$984,099$
			12 bars	$3''9$	$3''$	$2,179,359$
			16 bars	$6''6$	$5''$	$3,939,752$
			32 bars	$27''$	$24''$	$15,668,776$

Performance is reported on Table 1. The **Parsing** time is linear in the number of notes and chords in the corpus. In practice, this is a linear function of the total number of bars in the corpus. The **Training** time indicates the time needed to build and train the Markov+Meter models and the harmonic models and to initialise the belief propagation algorithms. The time reported in column **Next sol.** is less than the whole training time as the chord model is not retrained. The generation times increase quadratically with the length of the generated lead sheet. This experimental observation is consistent with the expected complexity of the algorithm (see Sect. 2.3) and is reflected in the size of the models.

The reported performance shows that generation becomes slow for lead sheets longer than 16 bars, especially with a large corpus, such as American Songwriters. Parsing the corpus uses non-optimised code, and its performance will be reduced by a substantial amount in the next version. The time needed to train the model and to find solutions may also be reduced by discarding very small probabilities in the model.

In general, music composed using statistical-based approaches is locally consistent but lacks a *sense of direction*, or *global structure* [6]. Consequently, purely autonomous generation is usually used to produce short musical sequences to be used as fragments in a longer piece. We observed that composers using the system rarely generate sequences exceeding 8 bars (see Sect. 4.3). The time taken to generate long lead sheets is therefore not a strong limitation of the system. The automatic generation of interesting long sequences requires models of large-scale musical structure including repetition of patterns, variations, and sections.

4.2 Harmonisation

FlowComposer may be used to *harmonise*, i.e. infer chord labels for a given melody in an imposed style. We illustrate such style-based harmonization by using FlowComposer to reharmonize "Yesterday", by the Beatles, in four styles: the Beatles themselves, Cole Porter, Michel Legrand, and Bill Evans.

Table 2 shows the number of songs in the corpus corresponding to each of these styles. Note that in the case of the Beatles, we did not include "Yesterday" in the corpus. The first 15 bars of the original harmonisation is shown on Fig. 5.

The re-harmonisation in the style of the Beatles (Fig. 6) uses many chords appearing in the original lead sheet, e.g., Dm, Gm, B♭. Some chords are simple harmonic substitutions, such as the Gm in place of B♭ on bar 4. The A7 on bar 7, which is not a substitution of the original harmony B♭-F, is quite surprising given the F in the melody. However such an augmented fifth is not unusual in the Beatles, and occurs for example in songs "I Want to Tell You" and "I'm Only Sleeping", in this latter case with the same resolution to a Dm chord. The progression in bars 13 to 14 is equal to the original. Overall, this re-harmonisation is probably less interesting than the original one, but is new, valid, and can be considered as in the style of the Beatles.

The harmonisation in Cole Porter's style (Fig. 7) uses an Eø7 (half diminished seventh) and an F♯o7 (diminished seventh) chords. Cole Porter is the composer who uses these chord types the most in the database. The progression DmM7-Dm-G7 appears identically in Cole Porter song "Do I Love You".

Fig. 5. The lead sheet of the first 15 bars of "Yesterday" with the original chord labels

Fig. 6. The lead sheet of the first 15 bars of Yesterday with an alternative harmonisation generated by FlowComposer in the style of the Beatles

Fig. 7. The lead sheet of the first 15 bars of Yesterday with a harmonisation generated in the style of the Cole Porter

Fig. 8. The lead sheet of the first 15 bars of Yesterday with a harmonisation generated in the style of the Michel Legrand

Fig. 9. The lead sheet of the first 15 bars of Yesterday with a harmonisation generated in the style of the Bill Evans

The use of a suspended 7th chord, E7sus in bar 10 in the example in Fig. 8 is typical of Michel Legrand: he is, in our database, the composer using them the most. The progression B half-diminished 7, Bø7, followed by E7sus actually occurs in "The Easy Way", or, transposed, in "Papa Can You Hear Me?". The E♭7 chord in bar 12 is a tritone substitution of the original A7 chord.

In the re-harmonisation in the style of Bill Evans (Fig. 9), the opening transition from E♭69 to A+7 (augmented 7th chord) appears in "Yet Neer Broken". Bill Evans is also the composer using 69 and 7♭9sus chords the most.

We encourage the reader to listen to these examples[1] to get a feel of the stylistic differences of the various harmonisations of the system.

---

[1] All examples are available online at: http://www.flow-machines.com/generation-of-lead-sheets-with-flowcomposer.

Table 2. Corpus size for several styles and execution times to harmonise the first 15 bars of "Yesterday". The total time is the time needed to parse the corpus and to train the model. Solutions are obtained virtually instantaneously by sampling the model

Style	Corpus size	Parsing time	Training time	Time next sol.	Total time
The Beatles	45 songs	85 ms	830 ms	15 ms	1″030
Cole Porter	70 songs	203 ms	1″568	15 ms	1″786
Michel Legrand	62 songs	181 ms	2″339	15 ms	2″535
Bill Evans	87 songs	132 ms	2″700	15 ms	2″847

Execution times are reported on Table 2. Most of the time is spent training the models, and is done only once, thanks to the persistence of user sessions. The third column reports the total time to get the first solution. Once the model is trained, it may be sampled virtually instantaneously to produce many representative solutions.

4.3 Interactive Composition

FlowComposer is not only an automatic generator but also a fully-fledged interactive composition tool for professional composers who use it as an active, create software collaborator. FlowComposer is integrated with an online lead sheet editor so that the system never gets in the way of the user's intentions and composition habits.

Fig. 10. Selection of two bars in the course of the composition of the song of Fig. 11; the two selected bars will be replaced by new musical material generated by FlowComposer

The general idea is that the user is responsible for creating the structure of the lead sheet and FlowComposer is used to generate music for user selected parts. The general structure of a lead sheet consists of a sequence of sections, which are played in a sequence with optional repetitions. A typical structure is AABA followed by a Coda, such as in "Yesterday". The editor allows the user to select contiguous fragments of the lead sheet and FlowComposer generates new music for this fragment; the fragments may contain chord labels, notes, or both (see Fig. 10).

The selection is not considered as an isolated musical fragment, but rather as belonging to the context of the current lead sheet. The non-selected parts are

Fig. 11. A song composed by French pop song writer Benoît Carré with FlowComposer

Fig. 12. The control panel with fields to select composition style and sliders to set *harmonisation* conformance, *inspiration*, and average *note duration* and *chord changes*

fixed and imposed as constraints to the system. Technically, the model covers the user selection extended to the music immediately before and after, to ensure that transitions between the selection and the surrounding musical context are in the chosen style.

The front end of the application provides the user with control on the generation parameters, which can be changed at any time: the training corpus, the number of notes or chord changes, and the harmonic conformance α (Fig. 12).

The system updates the model with the music entered in the lead sheet being composed. An additional control, called 'inspiration', is used to control the relative weight in the model of the training corpus and of the current composition. This control is typically used to put the emphasis on the current composition when the lead sheet contains already several bars of music to increase the probability that the music generated by the system will repeat some fragments present in the non-selected parts of the score.

French songwriter Benoît Carré is using FlowComposer for a forthcoming pop music album (see excerpts in Fig. 11). FlowComposer was also used by

Fig. 13. Final sheet of a song composed by Nathan Taylor and Benjamin Till using FlowComposer, as part of the "Beyond the Fence" musical

Fig. 14. A lead sheet composed by the authors of this article with FlowComposer, in the style of Miles Davis, and evaluated (informally) by fellow jazz musicians as particularly good

professional composers Nathan Taylor and Benjamin Till to compose three songs for the "Beyond the Fence" musical. For example, the chorus and some chord progression of "Scratch That Itch" come from FlowComposer. The final score has then been reworked by the composers. "Beyond the Fence" was the first musical ever created by software (not only songs, but also lyrics and the story line itself), and was performed at the Arts Theatre in London in February 2016 (see [5] for discussions about this pioneering experiment). Figure 13 shows the beginning of one of the songs in its final version. Figure 14 shows a lead sheet composed by the authors of this paper, using FlowComposer.

5   Conclusion

We presented FlowComposer, an online application for assisted music composition. The application enables users to compose musical lead sheets from partial information, and uses a generation algorithm to fill in the missing parts in the style of a chosen composer. The generation algorithm exploits a representation of meter as a REGULAR constraint which enables efficient sampling. Lead sheets are represented as a couple of meter-constrained Markov chains, synchronised through a likelihood function learnt from the selected corpus of lead sheets.

We have shown several typical uses of FlowComposer to generate lead sheets from scratch, from partial information, or to reharmonize existing melodies[2]. More work is being done to improve the user experience, notably by storing statistical models to avoid on-the-fly training, which calls for an incremental updating of these models.

FlowComposer can be seen as a successful example of an online application mixing statistical models with hard constraints. FlowComposer integrates a number of recent results in constraint programming and sampling, and has been used successfully in several professional music projects. We believe such a tool offers unprecedented value to users wanting meaningful assistance in composition, thanks to the powerful underlying style modeling approach. We also

---

[2] All the examples presented in this article are available online at http://www.flow-machines.com/generation-of-lead-sheets-with-flowcomposer.

believe that online applications mixing statistical models and hard constraints will be a very active thread of development for CP in the near future.

References

1. Anders, T., Miranda, E.R.: Constraint programming systems for modeling music theories and composition. ACM Comput. Surv. **43**(4), 30:1–30:38 (2011)
2. Begleiter, R., El-Yaniv, R., Yona, G.: On prediction using variable order markov models. J. Artif. Intell. Res. (JAIR) **22**, 385–421 (2004)
3. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription, pp. 1159–1166 (2012)
4. Brooks, F.P., Hopkins, A., Neumann, P.G., Wright, W.: An experiment in musical composition. IRE Trans. Electron. Comput. **3**, 175–182 (1957)
5. Colton, S., Llano, M.T., Hepworth, R., Charnley, J., Gale, C.V., Baron, A., Pachet, F., Roy, P., Gervás, P., Collins, N., Sturm, B., Weyde, T., Wolff, D., Lloyd, J.: The Beyond the Fence musical and computer says show documentary. In: 7th International Conference on Computational Creativity (ICCC 2016), Paris, France, June 2016
6. Eck, D., Schmidhuber, J.: Learning the long-term structure of the blues. In: Dorronsoro, J.R. (ed.) ICANN 2002. LNCS, vol. 2415, pp. 284–289. Springer, Heidelberg (2002)
7. Martín, D., Neullas, T., Pachet, F.: LeadsheetJS: a Javascript library for online lead sheet editing. In: 1st International Conference on Technologies for Music Notation and Representation (TENOR 2015), Paris, France, May 2015
8. Nierhaus, G.: Algorithmic Composition: Paradigms of Automated Music Generation. Springer, New York (2009)
9. Pachet, F., Roy, P., Barbieri, G.: Finite-length Markov processes with constraints. In: IJCAI, pp. 635–642 (2011)
10. Pachet, F., Suzda, J., Martinez, D.: A comprehensive online database of machine-readable lead-sheets for jazz standards. In: de Souza Britto Jr., A., Gouyon, F., Dixon, S.(ed.) ISMIR, pp. 275–280 (2013)
11. Pachet, F.: Flow-Machines: CP techniques to model style in music and text. ACP (Association for Constraint Programming) (2015). http://www.a4cp.org/node/1066
12. Papadopoulos, A., Pachet, F., Roy, P., Sakellariou, J.: Exact sampling for regular and Markov constraints with belief propagation. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 341–350. Springer, Heidelberg (2015)
13. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 482–495. Springer, Heidelberg (2004)

14. Rhodes, C., Lewis, D., Müllensiefen, D.: Bayesian model selection for harmonic labelling. In: Klouche, T., Noll, T. (eds.) MCM 2007. CCIS, vol. 37, pp. 107–116. Springer, Heidelberg (2009)
15. Roy, P., Pachet, F.: Enforcing meter in finite-length Markov sequences. In: des Jardins, M., Littman, M.L. (ed.) AAAI. AAAI Press (2013)

Enforcing Structure on Temporal Sequences: The Allen Constraint

Pierre Roy[(✉)], Guillaume Perez, Jean-Charles Régin,
Alexandre Papadopoulos, François Pachet, and Marco Marchini

Sony CSL Paris, 6, rue Amyot, 75005 Paris, France
`roypie@gmail.com`

Abstract. Recent applications of constraint programming to entertainment, *e.g.,* music or video, call for global constraints describing the structure of temporal sequences. A typical constraint approach is to model each temporal event in the sequence with one variable, and to state constraints on these indexed variables. However, this approach hampers the statement of constraints involving events based on temporal position, since the position depends on preceding events rather than on the index. We introduce Allen, a global constraint relating event indexes with temporal positions. Allen maintains two set-variables: the set of events occurring at a position defined by an Allen relation, and the set of their indexes. These variables enable defining structural and temporal synchronization properties that cannot be stated on indexed variables. We show that a model based on a local scheduling approach does not solve the problem, even for very small instances, highlighting the need for complex filtering. We present a model that uses Multi-valued Decision Diagrams (MDDs) to compile the Allen constraint. We show that this model can be used to state and solve two complex musical tasks: audio track synchronization and musical score generation.

Keywords: Global constraints · Temporal sequences · Music · MDD

1 Introduction

Many difficult combinatorial problems consist in arranging sequences of events in time, subject to *horizontal* and *vertical* constraints. These constraints are expressed on the *temporal position* of events. Horizontal constraints relate events in the same sequence, but occurring at different positions. Vertical constraints relate events occurring simultaneously, *i.e.,* at the same position in different sequences. This is similar to shceduling problems, such as job-shop scheduling, in which tasks are performed on machines according to sequential and resource constraints. The combination of horizontal and vertical constraints make these problems extremely difficult to solve: the job-shop scheduling problem is notoriously among the hardest combinatorial problems.

A typical constraint programming approach to generating such sequences is to define a variable for each item of the sequence, and to post constraints on

these variables. Temporal sequences challenge this model, since the position of an event is determined by the duration of all the preceding events, and so is only weakly dependent on its index. It is therefore difficult, if not impossible, to express temporal properties using constraints on item variables.

This problem appears naturally in application domains related to entertainment [1–4]. Structural properties usually involve long-range dependencies between events. Deep learning approaches attempt precisely at capturing these dependencies in a statistical model, to reproduce them during classification or sampling. However, the representations of structure in statistical models is not explicit, making them inappropriate for specifying hard constraints on sequences. In the next section, we describe a typical example of a structural constraint occurring in the generation of lead sheets, a type of musical notation.

Lead Sheet Generation

A lead sheet is a representation of a musical piece commonly used in popular music and consisting of a melody with chord labels on top, as shown on Fig. 1 (extract of *A Day in the Life* by Lennon/McCartney).

An important aspect of this lead sheet is that melodic patterns are distributed according to a temporal structure. For example, the pattern of bars 1-2 is repeated at bars 5-6. This type of structures is commonplace in popular music. To generate lead sheets with a similar temporal structure similar, a standard CP approach is to define one variable per note. However, notes have a different duration: bar 1 contains eight short notes (including the rest) and bar 2 contains only one long note. Consequently, there is no direct correspondence between the index of a note and its temporal position. This makes it hard to post constraints stating that the first two bars should be repeated two bars later, regardless of their number of notes; or any other constraint of the same kind. In Sect. 6, we show that our approach yields a practical solution to this problem.

Fig. 1. The first 8 bars of *A Day in the Life* by John Lennon and Paul McCartney

Automatic Accompaniment Generation

We describe and evaluate our technical contribution on the generation of musical accompaniment from audio multitrack recordings. We chose this example as it has immediate applications for computer generated musical improvisation or accompaniment generation, as mentioned in Sect. 5.1.

The task consists in generating a new multi-track audio accompaniment by reusing an existing multi-track recording. The original tracks are segmented into

chunks, using an onset detector [5]. Then new tracks are generated by recombining chunks, using concatenative synthesis [6]. Chunks in the generated tracks may appear in a different order than in the original track and may be used any number of times. Such a scheme involves several types of constraints: On the one hand, we have to constrain each track to avoid awkward chunk transitions, by allowing transitions that are similar to the transitions in the training corpus. The similarity is measured using acoustic features, see Sect. 5.1. On the other hand, to prevent the tracks from "drifting" from one another, *e.g.,* one track becomes increasingly louder while another track fades out, we have to *synchronize* the tracks at regular points in time, for instance at the onset of every bar.

This problem raises the same issue as lead sheet generation. The chunks have different durations, therefore the index of a chunk and its temporal position are not directly depending on one another.

Approaches

In the examples above and, more generally, in many interactive or content generation applications, we need to specify sequences with structural properties that cannot be inferred using statistical models. In the first example, the temporal structure in a lead sheet is not a statistical property that can be inferred from a set of examples, but is rather explicitly imposed, for instance by a user.

Constraint programming provides an ideal way of enforcing structure on sequences. However, as we highlighted earlier, we cannot state structural constraints on events based on their index alone.

Adopting a position-based model, in which variables represent events of smaller, *atomic duration* whereby longer objects are made up of several consecutive variables, solves this issue. For a given total duration, a fixed number of variables are defined and therefore indexes correspond to temporal positions. This requires discretizing time into a grid of equal-duration slices, small enough so that all events are aligned with the grid. In this model, the number of variables is considerably larger than the number of events in the generated sequences: if durations are expressed as fractions of the longest event, the atomic duration decreases with the *least common multiple* of the denominators, whose growth is exponential [7]. Hence, the size of the grid may be exponentially smaller than the event lengths, creating an intractable number of variables. Moreover, the position-based model requires additional horizontal constraints to aggregate atomic events to form longer objects. These constraints are not easy to specify in general. This approach is therefore not applicable in many real problems.

Several frameworks using constraint propagation make inferences about temporal relations from a qualitative [8] or quantitative standpoint [9]. The computational efficiency of these approaches is very limited in the general case, but they offer a precise and powerful representation of relations between times events.

Our Contribution

Allen [8] introduced an algebra with 13 binary relations between time intervals for temporal reasoning. We use this algebra as a language to express temporal positions and introduce the ALLEN global constraint, which defines variables corresponding to a given Allen relation. Technically, for a given time interval t and a given Allen relation $\mathcal{R}$, ALLEN maintains two set-variables: the set of events and the set of variable indexes satisfying $\mathcal{R}$ for t. Temporal properties of the sequence are represented by constraints defined on these set-variables. We present two models implementing ALLEN: the first model is based on a classical scheduling approach and the second model uses Multi-valued Decision Diagrams (MDDs). We show that the MDD-based approach, which performs tighter pruning, is much more efficient on the multitrack generation problem. The MDD representation and the associated filtering procedures are complex, which is why we present the first, simpler model. The two approaches are also used to run experimental comparisons showing that the MDD approach is necessary to solve actual synchronization problems.

A constraint based on Allen's algebra [1] takes a set of tasks, a set of Allen relations, a set of intervals, and checks that every task satisfies at least one relation for one interval. They apply this work to the generation of video summaries. In their approach, the checks for every task are independent from one another. On the contrary, in our approach, we use ALLEN to enforce *explicit* structural temporal properties, defined by Allen relations. Moreover, we take all specified properties into account in a single, global constraint.

The METER constraint [10] provides control on the duration and on various temporal properties of sequences, but is limited to the definition of *unary* constraints on events defined by their temporal position. It does not provide actual variables representing these events, and cannot be used, for instance, to state equality or difference constraints between events. ALLEN generalizes METER by defining two additional set-variables that can be used to state arbitrary, *e.g.,* *binary*, constraints enforcing temporal structures on sequences.

ALLEN is designed to model and solve problems of temporal sequence synchronization and structure. Previous work, including our own on using REGULAR [11] for music generation, never addressed this aspect.

2   Constraining Contiguous Temporal Sequences (CTS)

A *temporal event* e is a symbol with a duration $d(e)$. A *contiguous temporal sequence*, *CTS* for short, is a finite sequence of temporal events $(e_1, \ldots, e_n)$. A CTS is basically a concatenation of events: two consecutive events in a CTS are considered contiguous. Therefore, for a CTS $S = (e_1, \ldots, e_n)$, the *duration* $d(S)$ of S is the sum of the duration of the events contained in S, defined by $d(S) = \sum_{i=1}^{n} d(e_i)$. The *absolute temporal position*, or *starting time* of an event e_p in S is defined by $s(e_p) = \sum_{i=1}^{p-1} d(e_i)$. Note that the absolute temporal position is not an intrinsic property of a temporal event, it is a property of a temporal

event with respect to a CTS. A same temporal event may appear several times in a same CTS at different starting times.

In this article, we consider only temporal events with integer duration and, therefore, we address CTS in which all events have integer temporal positions.

Given a set E of temporal events, a model for the generation of CTS is to represent a CTS containing n temporal events of E as a sequence of n constrained variables $(X_1, \ldots, X_n)$, each with domain $dom(X_i) = E$. With this model, it is easy to state constraints relating events based on their index in the sequence, such as $X_1 = X_n$, or $X_i \neq X_{i+1}$. However, the absolute temporal position of an event in a CTS is not directly related to its index as it depends on the duration of all preceding events. There is therefore no straightforward way of constraining the elements of the sequence based on their absolute temporal position.

3   The Global Allen Constraint

The idea behind the ALLEN constraint is to use Allen relations between temporal intervals to specify some temporal element(s) of a CTS (the 13 atomic relations of Allen are given on Table 1). Let S be a CTS $(e_1, \ldots e_n)$. An Allen relation $\mathcal{R}$ and a temporal interval t specify a subsequence of S. For instance, if $\mathcal{R}$ is **d**, *i.e.*, the relation "during", and $t = [a, b]$, then $\mathcal{R}$ and t specify the subsequence of S containing the events which start after a and end before b.

Let E be a set of temporal events and let $X_1, \ldots, X_n$ be n constrained variables, each with domain E. The X_is are the *sequence* variables. Let t be a temporal interval and let $\mathcal{R}$ be a relation of Allen between temporal intervals (see Table 1). Let $\mathcal{I}$ be a set-variable, with domain $\{1, \ldots, n\}$ and $\mathcal{E}$ be a set-variable with domain E. The ALLEN constraint

$$\text{ALLEN}_{\mathcal{R},t}(X_1, \ldots, X_n, \mathcal{I}, \mathcal{E}) \tag{1}$$

ensures that $\mathcal{I}$ contains the *indexes* of all sequence variables X_i belonging to the subsequence of $(X_1, \ldots, X_n)$ specified by Allen relation $\mathcal{R}$ and temporal interval t. Similarly, the constraint (1) ensures that $\mathcal{E}$ contains the *values* of all sequence variables X_i belonging to the subsequence of $(X_1, \ldots, X_n)$ specified by $\mathcal{R}$ and t.

The ALLEN constraint defined above is satisfied if and only if

$$\mathcal{I} = \{i \in \{1, \ldots, n\} \mid [s(X_i), s(X_{i+1})] \ \mathcal{R} \ t\} \text{ and } \mathcal{E} = \{X_i \mid i \in \mathcal{I}\}$$

For example, the melody on Fig. 1 contains 36 notes (including rests). We represent each note as a temporal event and the melody as a CTS $M = (n_1, \ldots, n_{36})$, where n_i is the i-th note of the melody. Writing events as (note, duration) ordered pairs, with duration 1 for eighth-notes, we have:

$$n_1 = (rest, 1); n_2 = (B4, 1); n_3 = (D5, 1); n_4 = (B4, 1); n_5 = (E5, 1);$$

$$n_6 = (B4, 1); n_7 = (D5, 1); n_8 = (E5, 1); n_9 = (B4, 8); \ldots$$

where $D4$ denotes pitch D and octave 4. Note that $n_2 = n_4 = n_6$, $n_3 = n_7$, and $n_5 = n_8$.

Table 1. The 13 atomic relations of Allen. The lower bound of a time interval t_i is denoted by t_i- and the upper bound by t_i+.

Relation	Symbol	Example	Semantics	Inverse
t_1 before t_2	$<$	t_1 t_2	$t_1+ < t_2-$	$>$
t_1 equal t_2	**eq**	$t_1\,t_2$	$t_1- = t_2-$ and $t_1+ = t_2+$	**eq**
t_1 meets t_2	**m**	t_1 t_2	$t_1+ = t_2-$	**mi**
t_1 overlaps t_2	**o**	t_1 t_2	$t_1- < t_2-$ and $t_2- < t_1+ < t_2+$	**oi**
t_1 during t_2	**d**	t_1 t_2	$t_1- > t_2-$ and $t_1+ < t_2+$	**di**
t_1 starts t_2	**s**	t_1 t_2	$t_1- = t_2-$ and $t_1+ < t_2+$	**si**
t_1 finishes t_2	**f**	t_2 t_1	$t_1- > t_2-$ and $t_1+ = t_2+$	**fi**

Consider the relation of Allen "during" and the time interval defined by the first bar, *i.e.,* the interval starting at temporal position 1 and ending at temporal position 8 (as we count 1 for an eighth-note). The ALLEN constraint

$$\text{ALLEN}_{\mathbf{d}[1,8]}(n_1, \ldots, n_{36}, \mathcal{I}, \mathcal{E})$$

is satisfied if, and only if $\mathcal{I} = \{1, 2, \ldots, 8\}$ and $\mathcal{E} = \{(rest, 1), (B4, 1), (D5, 1), (E5, 1)\}$. Note that although the note $(B4, 1)$ appears three times in the first bar, it appears only once in $\mathcal{E}$, as temporal events do not have a starting time.

4   Implementing the Allen Constraint

We now describe two implementations of the ALLEN constraint. The first one is a simple model, based on scheduling, and performing only local propagations. We show in Sect. 5 that this model performs poorly. This justifies the need for a more elaborated model, using an MDD to represent the sequences explicitly, which makes it possible to prune more values during the search. This second model is presented in Sect. 4.2.

In both models, the sequence variables $X_1, \ldots, X_n$ take temporal event values.

4.1   A First Model

The ALLEN constraint can be seen as a non-preemptive scheduling problem with unary resources where variables correspond to activities having a variable duration. In this model, each variable X_i is associated with two variables S_i and D_i. Variable S_i represents the absolute temporal position of X_i in the CTS, and D_i represents the duration of X_i. The start and duration variables are related via a set of constraints

$$S_{i+1} = S_i + D_i, \forall i = 1, \ldots, n - 1 \tag{2}$$

with $S_1 = 0$.

In order to define the propagation rules, we will use the following five predicates:

– $\text{HOLDS}_{\mathcal{R},t}(s,d) \overset{def}{\iff} [s, s + d]\,\mathcal{R}\,t$, where s is a start time (i.e., absolute temporal position) and d a duration
– $\text{POSSIBLE}_{\mathcal{R},t}(i,e) \overset{def}{\iff} e \in dom(X_i)$ and $\exists s \in dom(S_i), \text{HOLDS}_{\mathcal{R},t}(s, d(e))$
– $\text{POSSIBLE}_{\mathcal{R},t}(i) \overset{def}{\iff} \exists e \in dom(X_i)$ such that $\text{POSSIBLE}_{\mathcal{R},t}(i,e)$
– $\text{REQUIRED}_{\mathcal{R},t}(i,e) \overset{def}{\iff} X_i = e$ and $\forall s \in dom(S_i), \text{HOLDS}_{\mathcal{R},t}(s, d(e))$
– $\text{REQUIRED}_{\mathcal{R},t}(i) \overset{def}{\iff} \forall e \in dom(X_i), \forall s \in dom(S_i), \text{HOLD}_{\mathcal{R},t}(s, d(e))$

Variables $\mathcal{I}$ and $\mathcal{E}$ are set-variables [12]. We will use the notation $lb(.)$ for the lower-bound of a set-variable domain and $ub(.)$ for its upper-bound. Intuitively, during the filtering procedure, the lower-bound $lb(\mathcal{I})$ (resp., $lb(\mathcal{E})$) is the set of required values for $\mathcal{I}$ (resp., $\mathcal{E}$). Similarly, the upper-bound $ub(\mathcal{I})$ (resp., $ub(\mathcal{E})$) is the set of possible values for $\mathcal{I}$ (resp., $\mathcal{E}$). The filtering rules presented below rely on the equivalences:

$$i \in ub(\mathcal{I}) \iff \text{POSSIBLE}_{\mathcal{R},t}(i) \tag{3}$$
$$i \in lb(\mathcal{I}) \iff \text{REQUIRED}_{\mathcal{R},t}(i) \tag{4}$$
$$e \in ub(\mathcal{E}) \iff \exists i, \text{POSSIBLE}_{\mathcal{R},t}(i,e) \tag{5}$$

Note that $e \in lb(\mathcal{E})$ is more difficult to express in terms of the predicates, which is why Rule (15) is more complex. In fact, reasoning on $lb(\mathcal{E})$ is the most complex operation for maintaining the consistency between the sequence variables and the set-variables. In the next section, we use an MDD model, which is sufficiently rich to infer the exact lower-bound $lb(\mathcal{E})$.

The consistency between the event, start, and duration variables, and the lower and upper bounds of the ALLEN set-variables, is maintained with a set of filtering rules.

When S_i is modified, *i.e.,* a value was removed from its domain, the following rules may apply:

$$i \in ub(\mathcal{I}) : \neg\text{POSSIBLE}_{\mathcal{R},t}(i) \Rightarrow i \notin ub(\mathcal{I})$$
$$\text{REQUIRED}_{\mathcal{R},t}(i) \Rightarrow i \in lb(\mathcal{I}) \tag{6}$$
$$i \in lb(\mathcal{I}) : e \in dom(X_i) \wedge (\forall s \in dom(S_i), \neg\text{HOLDS}_{\mathcal{R},t}(s, d(e)))$$
$$\Rightarrow e \notin dom(X_i) \tag{7}$$
$$i \notin ub(\mathcal{I}) : e \in dom(X_i) \wedge (\forall s \in dom(S_i), \text{HOLDS}_{\mathcal{R},t}(s, d(e)))$$
$$\Rightarrow e \notin dom(X_i) \tag{8}$$
$$e \in ub(\mathcal{E}) : \nexists j, \text{POSSIBLE}_{\mathcal{R},t}(j,e) \Rightarrow e \notin ub(\mathcal{E}) \tag{9}$$
$$e \notin ub(\mathcal{E}) : (\forall s \in dom(S_i), \text{HOLDS}_{\mathcal{R},t}(s, d(e)))$$
$$\Rightarrow e \notin dom(X_i) \tag{10}$$

Let us explain the first rule, Rule (6), in detail. Rule (6) is applied when a value is removed from the domain of S_i and if $i \in ub(\mathcal{I})$. The predicate $\text{POSSIBLE}_{\mathcal{R},t}(i)$ is evaluated, and if it does not hold true, index i is removed from $ub(\mathcal{I})$. The predicate $\text{REQUIRED}_{\mathcal{R},t}(i)$ is also evaluated, and if it holds true, index i is added to $lb(\mathcal{I})$. The variable S_i represents the starting times of the i-th event in the CTS. The property $i \in ub(\mathcal{I})$ means exactly that predicate $\text{POSSIBLE}_{\mathcal{R},t}(i)$ holds true (by Equivalence (3)). A possible consequence of removing a value from the domain of S_i is that there may be no more starting time s in S_i such that $[s, s + d(e)] \mathrel{\mathcal{R}} t$. Therefore, we reevaluate $\text{POSSIBLE}_{\mathcal{R},t}(i)$, and if it does not hold true anymore, we remove i from $ub(\mathcal{I})$. Another possible consequence of removing a value from S_i is that all remaining values $s \in dom(S_i)$ are such that $[s, s + d(e)] \mathrel{\mathcal{R}} t$ for any event $e \in dom(X_i)$, which means that $\text{REQUIRED}_{\mathcal{R},t}(i)$ holds true. Equivalence (4), we add index i to $lb(\mathcal{I})$.

When X_i is modified, we apply the following rules:

$$i \in ub(\mathcal{I}) : \neg\text{POSSIBLE}_{\mathcal{R},t}(i) \Rightarrow i \notin ub(\mathcal{I})$$
$$\text{REQUIRED}_{\mathcal{R},t}(i) \Rightarrow i \in lb(\mathcal{I}) \tag{11}$$
$$i \in lb(\mathcal{I}) : dom(X_i) = \{e\} \Rightarrow e \in lb(\mathcal{E})$$
$$s \in dom(S_i) \wedge (\forall e \in dom(X_i), \neg\text{HOLDS}_{\mathcal{R},t}(s, d(e)))$$
$$\Rightarrow s \notin dom(S_i) \tag{12}$$
$$i \notin ub(\mathcal{I}) : s \in dom(X_i) \wedge (\forall e \in dom(X_i), \text{HOLDS}_{\mathcal{R},t}(s, d(e)))$$
$$\Rightarrow s \notin dom(S_i) \tag{13}$$
$$e \in ub(\mathcal{E}) : \not\exists j, \text{POSSIBLE}_{\mathcal{R},t}(j, e) \Rightarrow e \notin ub(\mathcal{E}) \tag{14}$$
$$e \in lb(\mathcal{E}) : \exists i \in ub(\mathcal{I}) \text{ s.t.}$$
$$\text{POSSIBLE}_{\mathcal{R},t}(i, e) \wedge$$
$$\forall j \in ub(\mathcal{I}) \text{ s.t. } j \neq i, (e \notin dom(X_j) \vee \neg\text{POSSIBLE}_{\mathcal{R},t}(j, e))$$
$$\Rightarrow dom(X_i) = \{e\} \wedge i \in lb(\mathcal{I}) \tag{15}$$

When $\mathcal{I}$ is modified: if $i \in lb(\mathcal{I})$, apply Rule (7) and Rule (12); if $i \notin ub(\mathcal{I})$ apply Rule (8) and Rule (13). When $\mathcal{E}$ is modified: if $e \in lb(\mathcal{E})$ apply Rule (15); if $e \notin ub(\mathcal{E})$, $\forall i \in ub(\mathcal{I})$, apply Rule (10).

Most of those rules are straightforward implications of the predicate definitions, except rule (15). The first line of Rule (15) says that it is possible to have value e in the sequence. The following lines express the fact that if a only one variable X_i may take value e, we perform the assignment $X_i \leftarrow e$. We can easily verify that no rule removes any consistent value, *i.e.,* the rules are sound. However, this model does not remove all inconsistent values, *i.e.,* it does not achieve arc-consistency for ALLEN.

4.2   MDD-Based Model

This model uses an MDD constraint to represent the extension of the ALLEN constraint. By using propagators for MDDs, we can therefore achieve arc-consistency

of the whole ALLEN constraint. In fact, we show that we can even combine several ALLEN constraints into a single MDD and thus achieve arc-consistency for a set of ALLEN constraints. An MDD, in constraint programming, is a directed acyclic graph with one layer of nodes per constrained variable plus a final layer containing the single *true* node [13]. Defining the Allen constraint with MDDs can be decomposed into two steps:

– We first represent temporal constraint by a transition function computing the set of all temporal positions reachable from a given temporal position. The MDD is defined, starting from a root node associated to position 0, by successive applications of the transition function to determine all reachable temporal positions. An arc is associated to a duration, *i.e.,* time difference between the temporal position of its ending node and its origin node, *i.e.,* for and arc $a = (i, j)$, we have $t(j) = t(i) + d(a)$, where $t(.)$ denotes the temporal position of a node. The MDD constructed this way simply represents a sum function. We use the MDD Pattern construction defined in [14], which allow us to build an MDD based on a function of the node. Here the function node is the transition function between durations. Events are introduced in the MDD as follows: for each arc associated to a duration, we create as many arcs as there are events with this duration. Each arc in the resulting MDD is therefore labelled with a couple (event, duration).
– Then, for a given Allen relation, we identify all the arcs in the MDD that satisfy this relation. We can do this by noting that an arc $a = (i, j)$ occupies the temporal interval $[t(i), t(j)]$. These are the red arcs in Fig. 2.
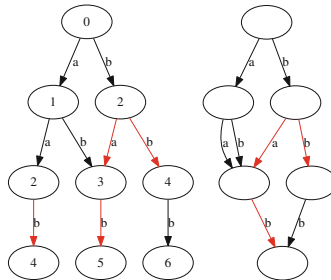
Fig. 2. The graph (left) and MDD (right) representations of the constraint ALLEN$_{\mathbf{d} \vee \mathbf{s} \vee \mathbf{fi} \vee \mathbf{eq}[2,5]}$ (see Table 2). Red labels correspond to values satisfying the constraint. Numbers in the graph on the left represent the temporal position. (Color figure online)

To illustrate this with an example, consider two events a and b with $d(a) = 1$ and $d(b) = 2$ and a sequence of three variables X_1, X_2, X_3 with domains $dom(X_1) = dom(X_2) = \{a, b\}$ and $dom(X_3) = \{b\}$. Let R denote the relation $\mathbf{d} \vee \mathbf{s} \vee \mathbf{f} \vee \mathbf{eq}$, which is similar to $\mathbf{d}$ except it is not strict. The extension of

Table 2. The extension of $\text{ALLEN}_{R[2,5]}$ for the example. Events that are, not strictly, during [2, 5] are in red.

X_1	X_2	X_3	$\mathcal{I}$	$\mathcal{E}$
a	a	b	$\{3\}$	$\{b\}$
a	b	b	$\{3\}$	$\{b\}$
b	a	b	$\{2,3\}$	$\{a,b\}$
b	b	b	$\{2\}$	$\{b\}$

$\text{ALLEN}_{R[2,5]}([X_1, X_2, X_3], \mathcal{I}, \mathcal{E})$ is shown in Table 2, where events that occur, not strictly, during [2,5] are in red.

The list of valid sequences of the constraint may be represented by the graph in Fig. 2 (left). Each layer represents one sequence variable (X_1 is the top layer, X_2 is the middle layer, and X_3 the bottom layer). Node labels represent start times and edge labels are events. Edges corresponding to events satisfying $\text{ALLEN}_{R[2,5]}$ are in red.

Note that the Allen relation does not change during search. As a consequence, one can ignore the temporal information in the nodes and apply the MDD reduction operation to the graph. This yields the reduced MDD in Fig. 2 (right). Note that the reduction distinguishes between black and red labels.

The algorithm presented in [15] is used to filter the domains of the sequence variables in the constraint represented by the MDD. The set-variables $\mathcal{I}$ and $\mathcal{E}$ must satisfy the following properties:

1. if $\forall a \in A_i$, a is red, then $i \in lb(\mathcal{I})$;
2. if $\exists a \in A_i$ such that a is red, then $i \in ub(\mathcal{I})$ and $label(a) \in ub(\mathcal{E})$.

where A_i denotes the i-th layer of the MDD, and a denotes an arc. By using the arcs deleted from the MDD, we maintain these properties incrementally.

When an event e is added to $lb(\mathcal{E})$, it means that all complete path in the MDD contains at least one red arc labelled with e. We modify the current MDD, notated MDDc, to integrate this new information. This is done by generating a new MDD, MDD(-e) containing all paths of MDDc that do not go through a red arc labelled with e. Then, we substract MDD(-e) from MDDc. This is efficient as MDD(-e) is a subgraph of MDDc. To generate MDD(-e), we duplicate MDDc, suppress all red arcs labelled with e, and propagate the suppression until every node belongs to a complete path (from the root node to a terminal node). This procedure is described in an article by Perez and Régin [16]. The complexity of these filtering operations is bounded by the size of the MDD.

In practice, the MDD representation is efficient because the bottom layers are highly compressed. This approach solves problems with up to 150 variables, which is enough for the targeted applications (see Sect. 5.3).

An important aspect of this approach it that we represent *several* ALLEN constraints stated on the same sequence in a *single* MDD. Then, we implement channeling relations between the MDD and the set-variables for each relation.

This allows us to achieve arc-consistent of the conjunction of all the ALLEN constraints. Note that integrating the set-variables in the MDD would require the definition of one MDD per Allen relation, and would sacrifice compression without improving filtering.

5   Evaluation

We evaluated the two models on an instance of the audio multitrack synchronization problem described in the introduction. The evaluation was implemented using the OR Tools[1] solver. Benchmark data will be made available online.

5.1   Description of the Benchmark

We use a three-track recording (guitar, bass, and drum) of the first 32 bars of song *Prayer in C* (Lilly Wood & The Prick). Each track is segmented using standard onset detection and quantized to $1/24^{th}$ of a beat (see Fig. 3).
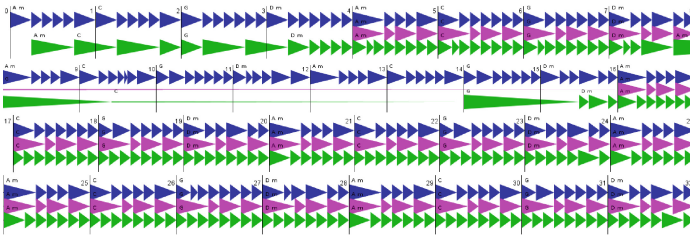
Fig. 3. A graphical representation of the guitar (top), bass (center), and drum (bottom) tracks of *Prayer in C*. Each track contains 32 bars and each triangle represents a chunk. Vertical lines indicate bar separations.

Chunks are categorized into *clusters* according to harmonic similarity (for pitched instruments) and timbre similarity for drums. The timbre is represented by Mel Frequency Cepstral Coefficients (MFCC) with 13 coefficients; the harmonic similarity is computed using the Harmonic Pitch-Class Profile (HPCP) with 36 divisions of the octave [17].

The guitar track contains 128 chunks with duration ranging from an eighth-note (half a beat) to a dotted quarter-note (1.5 beats), categorized into 13 harmonic clusters. The bass track contains 81 chunks (duration from half a beat to 1.5 beats) categorized into 9 bass clusters (harmonic similarity). The drum track contains 94 chunks with duration from half a beat to 20/3 beats, that is a full bar plus two thirds of a bar. There are 40 timbre-based drum clusters.

We state the problem of creating new multitracks as the generation of three sequences of chunks, each with an imposed total duration of n bars. Each bar

---

[1] OR Tools is open source and available at https://github.com/google/or-tools.

has four beats, and the duration of the shortest chunks is $^1/_8$ of a bar, therefore each sequence contains at most $p = 32n$ chunks.

We define a sequence of p chunk variables: $G_1, \ldots, G_p$ (guitar), $B_1, \ldots, B_p$ (bass), and $D_1, \ldots, D_p$ (drums). The domain of each variable is the set of chunks in the corresponding recorded track, plus a dummy chunk with duration 0, called the *padding element*, which we explain below.

For each track, all chunk transitions, *e.g.*, $G_i \to G_{i+1}$, are such that the associated cluster transition exists in the original track. Additionally, we synchronize the tracks together at the beginning of every bar. More precisely, let G_i, B_j, D_k be the three chunks playing at the beginning of bar b, and $C(G_I), C(B_j), C(D_k)$ be the corresponding clusters. We enforce that the same cluster "signature" exists somewhere in the original multitrack, not necessarily at the beginning of a bar. The underlying idea is that the cluster signatures of the original track are musically acceptable. Intuitively, this constraint imposes that the generated multitrack uses acceptable chunk signatures at the beginning of every bar but can "invent" new cluster signatures (new sounds) between bar lines.

It is easy to impose the total duration of $4n$ to each track by simply removing all nodes of the graph (see Fig. 2, left) whose label is greater than $4n$ and every node of the final layer whose label is different from $4n$. Every node with label $4n$ that is not in the final layer receives a new arc, labeled with the padding element, going to the next layer to a new node with the same label $4n$, since the padding element has a 0 duration. We repeat this process to the final layer. This allows us to generate sequences with fewer than p "actual" variables, the padding value being assigned to the "extra" variables.

The variables are subject to the binary constraints on chunk cluster transitions. These constraints are expressed as simple table constraints between consecutive chunk variables in each track. For example, $(C(G_i) \to C(G_{i+1})) \in \mathcal{C}_g$, where $\mathcal{C}_g$ is the set of all cluster transitions in the original guitar track. The same applies to the two other instruments.

The vertical synchronization constraints are represented by an ALLEN constraint for each track and for each bar. To specify the events that are playing at the beginning of bar i, we use the Allen relation $\mathbf{o} \vee \mathbf{s}$ applied to the time interval $[4(i-1), +\infty)$. In our context, this relation, one of the 2^{13} combinations of Allen relations, specifies exactly the intervals which "contain" the temporal point $4(i-1)$, the onset of bar i.

The ALLEN constraints for the guitar track are

$$\text{ALLEN}_{\mathbf{o} \vee \mathbf{s} \, [4(i-1), \infty)}([C_1^g, \ldots, C_p^g], \mathcal{I}_i^g, \mathcal{E}_i^g)$$

where C_i^g is the variable cluster(G_i). The synchronization itself is enforced by an *ad hoc* table constraint between $\mathcal{E}_i^g$, $\mathcal{E}_i^b$, and $\mathcal{E}_i^d$, where accepted triplets are cluster signatures of the original multitrack.

This approach applies to the generation of automatic accompaniment of an imposed melody in a given style. A demo video[2] is available online. Note that

---

[2] Video available online, https://www.youtube.com/watch?v=buXqNqBFd6E, examples at seconds 140, 176, and 216.

in this case, we enforce additional harmonic constraints to match a target chord sequence, in the case of the video above, the *Ode to joy.*

5.2   Evaluation of the First Model

We evaluate two implementations of the scheduling model, depending on how we implement constraint (2), which links start times and durations (defined in Sect. 4.1). First, we enforce arc-consistency on this ternary sum constraint. The model solves the problem for two bars in 8.4 seconds. It does not solve the problem for more than two bars in less that 30 min, which we consider a timeout. It is interesting to observe that the set of ternary sum constraints models a REGULAR constraint enforcing the graph on Fig. 2 (left). Furthermore, by enforcing AC for the ternary sum constraints, we also achieve AC for this REGULAR, since we obtain a model equivalent to the Berge acyclic decomposition of REGULAR [18]. The more complex MDD-based model differs from this approach by the compression applied to this graph, and its exploitation to obtain a tighter filtering on the set-variables.

We also implemented a lighter version where the ternary sums constraints only perform bound-consistency, based on the intuition that propagating information about the bounds of event duration offers a good trade off between simplicity and pruning. This model solves the problem for two bars in 5.4 seconds, but does not scale either to larger instances.

5.3   Evaluation of the MDD-Based Model

We use MDD4R [15] to perform the operations on the MDD constraint representing each ALLEN constraint. The code is implemented using the OR Tools solver. Note that, as said in Sect. 4.2, all the ALLEN constraints for a same track are represented by a single MDD.

The comparison with the performance of the simple model for ALLEN is clearly in favor of the MDD approach (see Table 3). The simple model does not solve problems longer than two bars in less than 30 min. In contrast, the MDD-based model solves the 14-bar problem in less than 2 min. The extra cost of

Table 3. The size of the MDDs and the execution time to find 5 solutions for various multitrack lengths

| n | MDD size (#Vertices, #Edges) | | | | | | Time |
	Guitar		Bass		Drum		(ms)
6	2382	41 k	848	13667	1864	73 k	2301
8	4199	74 k	1493	24 k	3817	156 k	7219
10	6530	117 k	2388	39 k	6513	275 k	23 k
12	9374	169 k	3623	61 k	9957	429 k	57 k
14	12 k	231 k	5085	87 k	14 k	617 k	112 k

performing the MDD construction and operations is more than compensated for by the higher pruning offered by this model, especially regarding the treatment of the set-variable $\mathcal{E}$.

6   Generation of Lead Sheets

To complete our presentation, we now address the problem of generating a melody, given a chord sequence, using ALLEN. This task, involving a single sequence, is a particular application of a general framework for lead sheet composition that we have developed. In this section, we sketch out the problem and how to state it using ALLEN. We do not report computation times as they vary considerably depending on the corpus and on the imposed chord sequence.

The melody is defined as a sequence of note variables, where a note is an event with a pitch and a duration, subject to constraints enforcing: (1) the duration of the melody matches that of the chord sequence, (2) the note transitions form acceptable musical intervals, (3) the notes match the current harmony (the chord), and (4) the melody satisfies an imposed pattern structure. We explain these constraints in detail on the example on Fig. 4.

Fig. 4. A 12-bar lead sheet generated using an ALLEN constraint.

For a 12-bar melody with a $^4/_4$ time signature, we define a sequence of note variables $[N] = [N_1, \ldots, N_p]$, with $p = 96$. The total duration is $48 = 12 \times 4$ beats. The melodic interval constraints are stated as binary table constraints between the pitch of any two consecutive notes. The allowed pitch intervals are collected from a corpus of 12000 lead sheets of popular music.

To enforce the harmonic constraints, we state an ALLEN constraint for each chord. For instance, on Fig. 4, the first chord, CM7, occupies the time interval [1,3] (half of the first bar). We define $\text{ALLEN}_{\mathbf{s}\vee\mathbf{d}\vee\mathbf{eq}\vee\mathbf{f}[1,3]}(\mathcal{E}_{\text{CM7}}, \mathcal{I}_{\text{CM7}})$, and use a unary constraint on $\mathcal{E}_{\text{CM7}}$, restricting its set-domain to the notes whose pitch is harmonically compatible with CM7. The allowed pitch for a given chord are extracted from our corpus.

Figure 4 shows a 12-bar lead sheet generated using an ALLEN constraint enforcing the equality between the patterns of bars 1-2 and bars 5-6, and between bars 3 and 9. Let P_1 denote the pattern of bars 1-2; P_2 that of bar 3; P_3 that of bars 5-6; and P_4 for bar 9. We state one ALLEN constraint for each pattern

- P_1 corresponds to $\text{ALLEN}_{\mathbf{s \vee eq \vee d \vee f}\,[1,9]}([N], \mathcal{E}_1, \mathcal{I}_1)$,
- P_3 corresponds to $\text{ALLEN}_{\mathbf{s \vee eq \vee d \vee f}\,[9,13]}([N], \mathcal{E}_2, \mathcal{I}_2)$,
- P_3 corresponds to $\text{ALLEN}_{\mathbf{s \vee eq \vee d \vee f}\,[17,25]}([N], \mathcal{E}_3, \mathcal{I}_3)$,
- P_4 corresponds to $\text{ALLEN}_{\mathbf{s \vee eq \vee d \vee f}\,[33,37]}([N], \mathcal{E}_4, \mathcal{I}_4)$.

However, the relations between patterns are relations between sub-sequences. They cannot be stated straightforwardly in terms of $\mathcal{E}_1$ and $\mathcal{E}_3$, as these do not maintain the order of the values.

For two index set-variables $\mathcal{I} = \{i_1, \ldots, i_k\}$ and $\mathcal{J} = \{j_1, \ldots, j_k\}$ with the indexes sorted by increasing order. We define the sub-sequence equality constraint

$$\text{SEQEQ}([X_1, \ldots, X_n], \mathcal{I}, \mathcal{J}) \iff X_{i_1} = X_{j_1} \wedge \cdots \wedge X_{i_k} = X_{j_k}$$

This constraint is expensive to filter as the domains of $\mathcal{I}$ and $\mathcal{J}$ are not known in advance. We add a redundant constraint to speed up the propagation:

$$\text{EQUAL}(\mathcal{E}, \mathcal{F})$$

where $\mathcal{E}$ and $\mathcal{F}$ are the even set-variables corresponding to $\mathcal{I}$ and $\mathcal{J}$ respectively. The filtering procedure for the equality constraint between set-variables (EQUAL) is quite standard; the filtering procedure for SEQEQ is not presented here. Basically, the filtering operation starts only once the first index in $\mathcal{I}$ and in $\mathcal{J}$ are known or, similarly, once the last index in $\mathcal{I}$ and in $\mathcal{J}$ are known.

7   Conclusion

We have presented the ALLEN global constraint. ALLEN maintains set-variables representing events in a temporal sequence in two ways: one variable is the set of events occurring at a given position, defined by an Allen relation with a reference time interval; the other variable is the set of indexes of these events. In practice, ALLEN offers the possibility to control the generation of temporal sequences by constraining events defined by their index *and* temporal position.

We proposed two models for ALLEN: a simple model using local propagation and a model based on MDDs and shown that the MDD representation, which achieves the global AC of the constraint, performs much better than the simple model on a temporal sequence synchronization problem.

ALLEN makes it possible to model and solve new types of problems involving structural constraints on patterns, represented by sub-sequences. We illustrated ALLEN on the task of synchronizing several audio tracks. Another application is the generation of structured lead sheets with pattern repetition. Such tasks could hardly be addressed using standard global constraints. More generally we believe that ALLEN addresses an increasing need for enforcing complex structural constraints in content generation for the entertainment domain.

References

1. Derrien, A., Fages, J.-G., Petit, T., Prud'homme, C.: A global constraint for a tractable class of temporal optimization problems. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 105–120. Springer, Heidelberg (2015)
2. Galvane, Q., Christie, M., Lino, C., Ronfard, R.: Camera-on-rails: automated computation of constrained camera paths. In: ACM SIGGRAPH Conference on Motion in Games, Paris, France, November 2015
3. Galvane, Q., Ronfard, R., Lino, C., Christie, M.: Continuity editing for 3D animation. In: AAAI Conference on Artificial Intelligence. AAAI Press, Austin, January 2015
4. Berrani, S.A., Boukadida, M.H., Gros, P.: Constraint satisfaction programming for video summarization. In: IEEE International Symposium on Multimedia, Anaheim, California, United States. IEEE, December 2013
5. Dixon, S.: Onset detection revisited. In: Proceedings of the 9th International Conference on Digital Audio Effects, Citeseer, vol. 120, pp. 133–137 (2006)
6. Maestre, E., Ramírez, R., Kersten, S., Serra, X.: Expressive concatenative synthesis by reusing samples from real performance recordings. Comput. Music J. **33**(4), 23–42 (2009)
7. Nair, M.: On chebyshev-type inequalities for primes. AMM **89**, 126–129 (1982)
8. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)
9. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. **49**(1–3), 61–95 (1991)
10. Roy, P., Pachet, F.: Enforcing meter in finite-length markov sequences. In: des Jardins, M., Littman, M.L. (eds.) AAAI. AAAI Press (2013)
11. Papadopoulos, A., Pachet, F., Roy, P., Sakellariou, J.: Exact sampling for regular and markov constraints with belief propagation. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 341–350. Springer, Heidelberg (2015)
12. Puget, J.F.: PECOS: a high level constraint programming language. In: Proceedings of Singapore International Conference on Intelligent Systems, SPICIS 1992, pp. 137–142 (1992)
13. Hoda, S., van Hoeve, W.-J., Hooker, J.N.: A systematic approach to MDD-based constraint programming. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 266–280. Springer, Heidelberg (2010)
14. Perez, G., Régin, J.C., Antipolis, U.N.S., Umr, I.S.: Efficient operations on MDDs for building constraint programming models. In: IJCAI International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, pp. 374–380 (2015)
15. Perez, G., Régin, J.-C.: Improving GAC-4 for table and MDD constraints. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 606–621. Springer, Heidelberg (2014)
16. Perez, G., Régin, J.C.: Relations between MDDs and Tuples and Dynamic Modifications of MDDs based constraints. arXiv preprint (2015). arXiv:1505.02552
17. Gómez, E.: Tonal Description of Music Audio Signals. Ph.D. thesis, Universitat Pompeu Fabra (2006)
18. Quimper, C.-G., Walsh, T.: Global grammar constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 751–755. Springer, Heidelberg (2006)

Constraint Programming Approach to the Problem of Generating Milton Babbitt's All-Partition Arrays

Tsubasa Tanaka[1(✉)], Brian Bemman[2], and David Meredith[2]

[1] STMS Lab: IRCAM, CNRS, UPMC, Paris, France
`tsubasa.tanaka@ircam.fr`
[2] Aalborg University, Aalborg, Denmark
`{bb,dave}@create.aau.dk`

Abstract. Milton Babbitt (1916–2011) was a composer of twelve-tone serial music noted for creating the *all-partition array*. One part of the problem in generating an all-partition array requires finding a covering of a pitch-class matrix by a collection of sets, each forming a region containing 12 distinct elements and corresponding to a distinct integer partition of 12. Constraint programming (CP) is a tool for solving such combinatorial and constraint satisfaction problems. In this paper, we use CP for the first time to formalize this problem in generating an all-partition array. Solving the whole of this problem is difficult and few known solutions exist. Therefore, we propose solving two sub-problems and joining these to form a complete solution. We conclude by presenting a solution found using this method. Our solution is the first we are aware of to be discovered automatically using a computer and differs from those found by composers.

Keywords: Babbitt · All-partition array · Computational musicology · Constraint programming

1 Introduction

Milton Babbitt (1916–2011) was a composer of twelve-tone serial music noted for developing highly constrained and often complex musical structures. Many of his pieces are organized according to one such structure known as the *all-partition array* [1]. An all-partition array is a covering of a matrix of pitch-class integers by a collection of sets, each of which forms a region in this matrix containing 12 distinct pitch classes from consecutive elements in its rows and that corresponds to a distinct integer partition of 12 (to be clarified in the next section). This unique structure imposes a strict organization on the pitch classes in his works, and it serves as both a method of musical composition and musical form. Moreover, the all-partition array allowed Babbitt one of many ways to achieve *maximal diversity* in his music.[1]

---

[1] Maximal diversity is the presentation of as many musical parameters in as many different ways as possible [2].

In this paper, we formulate one part of the problem in generating an all-partition array, beginning from a given matrix of pitch-class integers, using constraint programming (CP) and with a particular focus on its mathematical aspects. Using our model and a method for dividing this matrix into smaller, sub-problems, we obtained a solution, which, we believe, is the first to be discovered automatically using a computer and differs from those found by composers. CP is a programming paradigm that has been successfully applied to the solving of various constraint satisfaction problems in music [3–7]. It seems natural then, that CP could be used in the problem we address here. Moreover, having such a model could, for example, be used as a basis for generating new musical works.

1.1   The Structure of an All-Partition Array

In this section, we describe the structure of an all-partition array in a way that assumes the reader has a basic understanding of pitch class set theory. Constructing an all-partition array begins with the construction of an $I \times J$ matrix, A, whose elements are pitch-class integers, $0, 1, \ldots, 11$, where each row contains $J/12$ twelve-tone rows. The dimensions of this matrix constrain the most important requirement of the structure of an all-partition array, however, Babbitt generally limited himself to sizes of 4×96, 6×96, and 12×72 [2]. In this paper, we consider only matrices where $I = 4$ and $J = 96$, as matrices of this size figure prominently in Babbitt's music [2]. This results in a 4×96 matrix of pitch classes, containing 32 twelve-tone rows from the possible 48 related by any combination of transposition, inversion and retrograde (i.e., reversal). In other words, A will contain an approximately uniform distribution of 32 occurrences of each of the integers from 0 to 11.[2] On the musical surface, rows of this matrix become expressed as 'musical voices', typically distinguished from one another by instrumental register [2].

A complete all-partition array is a covering of matrix, A, by K sets, each of which is itself a partition of the set $\{0, 1, \ldots, 11\}$ whose parts (1) contain consecutive row elements from A and (2) have cardinalities equal to the summands in one of the K distinct integer partitions of 12 (e.g., $6 + 6$ or $5 + 4 + 2 + 1$) containing I or fewer summands greater than zero.[3] Figure 1 shows a 4×12 excerpt from a 4×96 pitch-class matrix, A, and two such sets forming *regions* in A each containing every pitch class exactly once and corresponding to two distinct integer partitions, whose exact "shapes" are more precisely represented as the *integer compositions*, $\mathrm{IntComp}_{12}(4, 4, 4, 0)$ and $\mathrm{IntComp}_{12}(0, 6, 3, 3)$.[4]

---

[2] For a more detailed description of the constraints governing the organization of matrices in Babbitt's music, see [2,8].

[3] We denote an integer partition of an integer, L, by $\mathrm{IntPart}_L(s_1, s_2, \ldots, s_I)$ and define it to be an ordered set of non-negative integers, $\langle s_1, s_2, \ldots, s_I \rangle$, where $L = \sum_{i=1}^{I} s_i$ and $s_1 \geq s_2 \geq \cdots \geq s_I$.

[4] We define an *integer composition* of a positive integer, L, denoted by $\mathrm{IntComp}_L(s_1, s_2, \ldots, s_I)$, to also be an ordered set of I non-negative integers, $\langle s_1, s_2, \ldots, s_I \rangle$, where $L = \sum_{i=1}^{I} s_i$.

Fig. 1. A 4×12 excerpt from a 4×96 pitch-class matrix with two distinct integer partition regions represented precisely by the integer compositions, $\mathrm{IntComp}_{12}(4, 4, 4, 0)$ (in dark gray) and $\mathrm{IntComp}_{12}(0, 6, 3, 3)$ (in light gray), each containing every pitch class exactly once.

Note, in Fig. 1, that each summand (from left to right) in $\mathrm{IntComp}_{12}(4, 4, 4, 0)$, gives the number of elements in the corresponding row of the matrix (from top to bottom) in this region. Unlike common tiling problems using, for example, polyominoes, these regions need not have connected interiors, as demonstrated by the second region in Fig. 1 between rows 3 and 4. On the musical surface, the distinct shape of each region helps contribute to a progression of 'musical voices' that vary in textural density, allowing for relatively thick textures in, e.g., $\mathrm{IntComp}_{12}(3, 3, 3, 3)$ (with four participating parts) and comparatively sparse textures in, e.g., $\mathrm{IntComp}_{12}(11, 0, 1, 0)$ (with two participating parts).

There exist a total of 34 distinct integer partitions of 12 into 4 or fewer non-zero summands [2]. An all-partition array with four rows will thus contain $K = 34$ regions, each containing every pitch class exactly once and each with a distinct "shape" determined by an integer composition defining a distinct integer partition. However, the number of pitch classes required to satisfy this constraint, $34 \times 12 = 408$, exceeds the size of a 4×96 matrix containing 384 elements, by 24. In order to satisfy this constraint, contiguous regions may share pitch classes, with the added constraint that only horizontal *overlaps* of at most one pitch class in each row are allowed for each of the 34 integer partition regions. Figure 2 shows a third region, $\mathrm{IntComp}_{12}(5, 1, 0, 6)$ (in medium gray), in the matrix shown in Fig. 1, where two of its elements result from overlapped pitch classes from previous regions. Note, in Fig. 2, the two horizontal overlaps of pitch class, 7 (in row 1 and belonging to the first region) and 8 (in row 4 and belonging to the second region), required to have each pitch class occur exactly once in the third integer partition region. This means that while contiguous regions may share pitch classes, such regions need not be necessarily adjacent in sequence.

Composers have primarily relied on constructing all-partition arrays by hand and at least some of their methods have been published [1,9,10]. Algorithms for automating this task have also been proposed [8,11]. However, generating an all-partition array is a large combinatorial problem and satisfying the constraints of its structure is difficult. To date, none of these algorithms have been able to solve this problem automatically. This observation motivates our decision here to look for alternative programming paradigms and methods for possibly better addressing this problem. In Sect. 2, we present our CP constraints for implementing the problem of generating an all-partition array from a given matrix. As solving for

11	10	3	7	6	2	4	5	8	1	9	0
8	9	4	0	1	5	3	2	11	6	10	7
2	5	1	6	9	10	0	8	7	11	4	3
7	4	8	3	0	11	9	1	2	10	5	6

Fig. 2. A 4×12 excerpt from a 4×96 pitch-class matrix with a third integer composition, $\mathrm{IntComp}_{12}(5, 1, 0, 6)$ (in medium gray), sharing one pitch class from each of the two previous regions.

the entire matrix directly is difficult, in Sect. 3, we present a method of dividing this matrix into two smaller matrices, choosing integer partitions based on how frequently they appear in solutions to one of these smaller matrices, and re-joining them to form a complete solution. We conclude here with a solution discovered using this method.

2   CP Constraints for the Problem of Generating an All-Partition Array from a Given Matrix

We begin the discussion of our CP constraints for generating an all-partition array, with a given matrix found in one of Babbitt's works based on the all-partition array.[5] Let $(A_{i,j})$ be this $(4, 96)$-matrix whose elements are the pitch-class integers, $0, 1, \ldots, 11$. We denote the number of rows and columns by I and J, respectively. Let $x_{i,j,k}$ $(1 \leq i \leq I, 1 \leq j \leq J)$ be a binary variable corresponding to each location (i, j) in A and a subset (i.e., a region) identified by the integer k, where $1 \leq k \leq K$ and $K = 34$. There are then 34 sets of 384 such variables. Each of these variables will indicate whether or not a location (i, j) belongs to a *candidate set*, which we denote, C_k, for the kth position in the sequence of 34 regions. For C_k to be a candidate set, it must form a region in A (as described in Sect. 1), by satisfying two conditions, *consecutiveness* and *containment*, which we will introduce below. Having satisfied these conditions, C_k will be a candidate set in a possible solution to our problem, in which its elements correspond to 12 distinct pitch classes in A and whose "shape" is defined by an integer composition. Additional constraints e.g., ensuring that each of these candidate sets is then a distinct integer partition and that their overlaps do not exceed one in each row, will then complete our formulation of this problem.

2.1   Consecutiveness

The condition of consecutiveness states that pitch classes belonging to the same region and row in A must lie adjacent to one another with no gaps between. We ensure this is the case by placing constraints on the strings of 0's and 1's that are allowed in the rows formed by $\langle x_{i,1,k}, x_{i,2,k}, \ldots, x_{i,J,k} \rangle$ for each (i, k). If, for

---

[5] Examples of this matrix can be found in Babbitt's *My Ends are My Beginnings* (1978) and *Beaten Paths* (1988), among others.

example, the string $\langle \ldots, 0, 1, \ldots \rangle$ appears in the ith row for some k, then there can be no 1 occurring before 0. This is expressed by the following:

$$\forall i \in [1, I], \forall j \in [3, J], \forall k \in [1, K],$$

$$(x_{i,j-1,k} = 0 \ \wedge \ x_{i,j,k} = 1) \implies \bigwedge_{j'=1}^{j-2}(x_{i,j',k} = 0). \tag{1}$$

On the other hand, if $\langle \ldots, 1, 0, \ldots \rangle$ appears in this row, then there can be no 1 after 0. This is expressed by the following:

$$\forall i \in [1, I], \forall j \in [1, J-2], \forall k \in [1, K],$$

$$(x_{i,j,k} = 1 \ \wedge \ x_{i,j+1,k} = 0) \implies \bigwedge_{j'=j+2}^{J}(x_{i,j',k} = 0). \tag{2}$$

In other words, all 1's in $\langle x_{i,1,k}, x_{i,2,k}, \ldots, x_{i,J,k} \rangle$ for each (i, k) must be consecutive, with any 0's lying to the left or right end points of this string.

2.2   Containment

The condition of containment states that regions in A must contain 12 distinct pitch classes. Let B_p $(0 \le p \le 11)$ be the set of all locations (i, j) of pitch class p in matrix A. From this, we can express the condition of containment by the following:

$$\forall p \in [0, 11], \forall k \in [1, K], \sum_{(i,j) \in B_p} x_{i,j,k} = 1, \tag{3}$$

where for each k, $x_{i,j,k}$ is equal to 1 at one and only one location (i, j) whose pitch class is p in A. When this is the case, C_k will contain one of each pitch class.

2.3   Covering All (i, j) in A

A solution to our problem requires that every one element in A is covered by at least one of the regions, C_k. We can express this condition by the following constraint:

$$\forall i \in [1, I], \forall j \in [1, J], \bigvee_{k=1}^{K}(x_{i,j,k} = 1). \tag{4}$$

2.4   Restrictions on the Left-to-right Order of Candidate Sets and Their Overlaps

As discussed in Sect. 1, adjacent regions need not be contiguous in each row in A, however, there are restrictions on their left-to-right order and allowed overlaps. The number of overlaps in each row between these regions must not exceed 1. We can express this restriction by the following constraint:

$$\forall i \in [1, I], \forall j \in [2, J], \forall k \in [1, K-1],$$

$$(x_{i,j,k} = 1) \implies \bigwedge_{k'=k+1}^{K}(x_{i,j-1,k'} = 0). \tag{5}$$

When combined with the constraint of consecutiveness, constraint 5 means that if $x_{i,j,k}$ is equal to 1, the ith row of $C_{k'}$, whose k' is greater than k, is either (1) located at the right-hand side of (i, j) without overlapping the ith row of C_k or (2) has only one overlap at the right-most element of the ith row of C_k.

2.5   Candidate Sets as All Different Integer Partitions

In order to determine that the integer composition "shape" of C_k is a distinct integer partition, we introduce two variables, $y_{i,k,l}$ and $z_{k,l}$. Let $y_{i,k,l}$ be a binary variable that indicates whether or not the length of the ith row of C_k is greater than or equal to l $(1 \leq l \leq L, L = 12)$, by introducing the following two constraints:

$$\forall i \in [1, I], \forall k \in [1, K], \ \sum_{j=1}^{J} x_{i,j,k} = \sum_{l=1}^{L} y_{i,k,l} \tag{6}$$

$$\forall i \in [1, I], \forall k \in [1, K], \forall l \in [2, L], \ (y_{i,k,l} = 1) \implies (y_{i,k,l-1} = 1). \tag{7}$$

Equation 6 states that the sum of all elements in $\langle y_{i,k,1}, y_{i,k,2}, \ldots, y_{i,k,L} \rangle$ is equal to the length of the ith row of C_k while Eq. 7 states that its elements equal to 1 begin in the first position and are consecutive. For example, when the length of the ith row of C_k is 3, $\langle y_{i,k,1}, y_{i,k,2}, \ldots, y_{i,k,L} \rangle$ is $\langle 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$. The total number of rows in C_k whose lengths are greater than or equal to l is given by $\sum_{i=1}^{I} y_{i,k,l}$. Let $z_{k,l}$ $(0 \leq z_{k,l} \leq I)$ be an integer variable that is equal to $\sum_{i=1}^{I} y_{i,k,l}$ $(1 \leq l \leq L)$ with the following constraint:

$$\forall k \in [1, K], \forall l \in [1, L], \ z_{k,l} = \sum_{i=1}^{I} y_{i,k,l}. \tag{8}$$

The ordered set of twelve values $z_{k,l}$ $(1 \leq l \leq L)$ will then identify the type of integer partition corresponding to C_k. For example, when $z_{k,l}$ is $\langle 4, 4, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0 \rangle$, C_k is $\text{IntPart}_{12}(5, 3, 2, 2)$. We denote this set $z_{k,l}$ corresponding to an integer partition n by $P_n = \langle P_{n,1}, P_{n,2}, \ldots P_{n,L} \rangle$ $(1 \leq n \leq N, N = 34)$, where integer partitions appear in reverse lexicographical order, meaning that those containing the fewest parts and largest part lengths appear first. For example, P_1 is $\text{IntPart}_{12}(12, 0, 0, 0)$ and P_{34} is $\text{IntPart}_{12}(3, 3, 3, 3)$. From this, we determine the integer composition shape of C_k to be the integer partition n by the following constraint:

$$\forall k \in [1, K], \forall n \in [1, N], \ (w_k = n) \iff \bigwedge_{l=1}^{L} (z_{k,l} = P_{n,l}), \tag{9}$$

where w_k $(1 \leq w_k \leq N)$ is an integer variable that indicates to which P_n C_k corresponds. We can now express the condition that all integer partitions are distinct by the constraint, $\text{ALLDIFFERENT}(w_1, w_2, \ldots, w_K)$.

3 Solution

In order to confirm that our formulation of this problem is accurate, we implemented our constraints described in Sect. 2 and supplied these to a CP solver (Sugar v2-1-0 [12,13]). We first tried to solve for the whole matrix directly, however, we were unable to obtain a solution after a day of calculation. We decided instead, to divide the matrix into two, equally-sized halves and try solving for each in such a way that their re-joining would form a complete solution to the original problem. We made this division of the original matrix at $[1,I]\times[1,J/2]$. Columns 1 to ($J/2$) then correspond to the first smaller matrix we denote by A_1 and columns ($J/2$)+1 to J correspond to the second smaller matrix we denote by A_2. We allocated $34/2 = 17$ integer partitions to be found in each.

With little modification, our constraints can be adapted to the solving of these sub-problems. These changes include modifying B_p (in Eq. 3) to contain only the locations of pitch classes in either A_1 or A_2, setting K to be the new number of partitions in each (i.e., 17) and J to be their new column lengths $96/2 = 48$. Solutions to A_1 and A_2 in which no integer partition is used more than once and contains only pitch classes from one or the other matrix (but not both), collectively form a solution to the original problem. Due to its smaller size, we were able to find solutions beginning with A_1 over the course of a day, in which 506 were found. Naturally, solving for A_1 makes finding a solution in A_2 more difficult as the number of available partitions is now fewer, and in fact, all 506 solutions to A_1 made A_2 unsatisfiable. We noticed, however, that certain

Table 1. A generated all-partition array corresponding to a complete solution to our problem, represented in the way used by music theorists [2]. Each column contains the elements in A belonging to C_k, where a dash indicates those that overlap. Note, that partitions are denoted using a shorthand notation, e.g., 4^3, where the base indicates the length of a part and the exponent denotes its number of occurrences. For clarity, the integers 10 and 11 have been replaced by the letters t and e, respectively.

4^3	62^3	641^2	82^2	93	91^3	543	831	3^4	$4^2 2^2$	$53^2 1$	84
et37	62	4581	90		7	t6e	23510498	-867	2t	e31	-1094
8940	15	32e6t7			859410t23		e67	-0	549	10	8te27365
2516	9t	0	87e4365t	219	e		03847		1t2	9653	784
	74830e	9	12	t56e07348	6		5291	t	03e	-e478	t6592

$5^2 2$	75	$43^2 2$	532^2	651	921	642	731^2	63^2	732	$10\,1^2$
85637		-72e	t8	01945	32		7et6890	514	-4e2	-2t36795481
		-58	49013	-3et276	450891et7	26	3		-30	-0
-4e09t	-t5126	430	7e	8	6	95t1		-124	0e3872 16t9578	e
-21	4380e79	-9t16	-625			304e87	5	9t6		(vacant)

741	12	6321	821^2	$10\,2$	5421	$5^2 1^2$	6^2	$72^2 1$	$4^2 31$	$11\,1$
023t67e			9		8504	1	2e3t76	48	9501	
9185	42673te10598	-e	67		-7	t3e29	-908145	73	26et	
4		0396t5	-5	21	e3	40785		619t20e	837	4
		127	83e0421t	-t5964738e0	2t916	-6			5	380e79t1625

partitions in these 506 solutions e.g., $\text{IntPart}_{12}(3,3,3,3)$ and $\text{IntPart}_{12}(4,3,3,2)$ occurred far less frequently than others. It would be reasonable then to conclude that solutions in A_1 which contain the greatest number of these less frequently occurring partitions will make solving for A_2 more likely, as the fewer available partitions in A_2 now consist of a proportionally greater number of frequently occurring partitions. Therefore, we solved again for A_1, this time by arbitrarily restricting the domain of w_k to exclude the top 6 most frequently occurring partitions and include the top 5 least frequently occurring partitions.

If we denote the subset of integers from $[1, 34]$ corresponding to the partitions found in this solution to A_1, S, then the domain of w_k for possible solutions to A_2 becomes $[1, 34] \setminus S$. We then tried solving for A_2, under the assumption that its proportionally greater number of more frequently occurring partitions would make finding a solution easier. While this means we exclude possible solutions e.g., ones in which a rarely occurring partition occurs in A_2 or where a partition contains pitch classes from both A_1 and A_2, we were able to generate a complete solution in this way. Solving for A_1 took approx. 4 minutes while solving for A_2 took approx. 28 minutes. Table 1 shows the complete solution found using this method of re-joining A_1 and A_2.

4   Conclusion

In this paper, we have introduced a novel formulation of one part of the problem of generating an all-partition array, beginning from a given matrix, using constraint programming (CP). Solving for the whole of this matrix directly proved too difficult using our constraints. Therefore, we introduced a method of dividing the matrix into two halves, solving for each and then re-joining them to form a complete solution. Using this method, we were able to discover a solution. This solution is the first we are aware of to be automatically generated by a computer. Moreover, it is an all-together new all-partition array from those previously discovered by Babbitt and other composers. In future work, we hope to examine in more detail how to make finding solutions in larger matrices possible and without excluding potential solutions.

References

1. Babbitt, M.: Since Schoenberg. Perspect. New Music **12**(1/2), 3–28 (1973)
2. Mead, A.: An Introduction to the Music of Milton Babbitt. Princeton University Press, Princeton (1994)

3. Anders, T., Anagnostopoulou, C., Alcorn, M.: Strasheela: design and usage of a music composition environment based on the Oz programming model. In: Van Roy, P. (ed.) MOZ 2004. LNCS, vol. 3389, pp. 277–291. Springer, Heidelberg (2005)

4. Laurson, M., Kuuskankare, M.: A constraint based approach to musical textures and instrumental writing. In: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming, Musical Constraints Workshop (2001)

5. Carpentier, G., Assayag, G., Saint-James, E.: Solving the musical orchestration problem using multiobjective constrained optimization with a genetic local search approach. Heuristics **16**(5), 681–714 (2010). Springer

6. Chemillier, M., Truchet, C.: Two musical CSPs. In: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming, Musical Constraints Workshop (2001)

7. Puget, J.F., Régin, J.C.: Solving the All Interval Problem. https://ianm.host.cs.st-andrews.ac.uk/CSPLib/prob/prob007/puget.pdf

8. Bemman, B., Meredith, D.: Generating Milton Babbitt's all-partition arrays. J. New Music Res. **45**(2), (2016a). http://www.tandfonline.com/doi/full/10.1080/09298215.2016.1172646

9. Starr, D., Morris, R.: A general theory of combinatoriality and the aggregate, part 1. Perspect. New Music **16**(1), 3–35 (1977)

10. Starr, D., Morris, R.: A general theory of combinatoriality and the aggregate, part 2. Perspect. New Music **16**(2), 50–84 (1978)

11. Bazelow, A.R., Brickle, F.: A combinatorial problem in music theory: Babbitt's partition problem (I). Ann. N. Y. Acad. Sci. **319**(1), 47–63 (1979)

12. http://bach.istc.kobe-u.ac.jp/sugar/

13. Naoyuki, T., Mutsunori, B.: Sugar: A CSP to SAT translator based on order encoding. In: Proceedings of the 2nd International CSP Solver Competition, pp. 65–69 (2008)

Preference, Social Choice
and Optimization Track

A Dynamic Programming-Based MCMC Framework for Solving DCOPs with GPUs

Ferdinando Fioretto[1,2(✉)], William Yeoh[1], and Enrico Pontelli[1]

[1] Department of Computer Science, New Mexico State University, Las Cruces, USA
{ffiorett,wyeoh,epontell}@cs.nmsu.edu
[2] Department of Mathematics and Computer Science,
University of Udine, Udine, Italy

Abstract. The field of *Distributed Constraint Optimization* (DCOP) has gained momentum in recent years, thanks to its ability to address various applications related to multi-agent coordination. Nevertheless, solving DCOPs is computationally challenging. Thus, in large scale, complex applications, incomplete DCOP algorithms are necessary. Recently, researchers have introduced a promising class of incomplete DCOP algorithms, based on *sampling*. However, this paradigm requires a multitude of samples to ensure convergence. This paper exploits the property that sampling is amenable to parallelization, and introduces a general framework, called *Distributed MCMC* (DMCMC), that is based on a dynamic programming procedure and uses *Markov Chain Monte Carlo* (MCMC) sampling algorithms to solve DCOPs. Additionally, DMCMC harnesses the parallel computing power of *Graphical Processing Units* (GPUs) to speed-up the sampling process. The experimental results show that DMCMC can find good solutions up to two order of magnitude faster than other incomplete DCOP algorithms.

1 Introduction

In a *Distributed Constraint Optimization Problem* (DCOP), multiple agents coordinate assignments of values to their variables to maximize the sum of the resulting constraint utilities [18,32]. DCOP is a powerful paradigm to describe and solve many practical problems in a variety of application domains, such as distributed scheduling, coordination of unmanned air vehicles, smart grid electrical networks, and sensor networks [10,24,28,34]. DCOP researchers have proposed a wide variety of solution approaches, from distributed search-based algorithms [15,18,31] to distributed inference-based algorithms [21,30], as well as solvers that use GPUs [3,4] and logic programming [12,13] formulations. Complete DCOP algorithms find optimal solutions at the cost of large runtimes, while

incomplete approaches trade optimality for faster execution. Since finding optimal DCOP solutions is NP-hard, incomplete algorithms are often necessary to solve larger problems. A further challenge to the applicability of DCOPs to more general classes of problems is the common assumption that each agent controls exactly one variable during problem resolution, which is often unrealistic. To cope with such restrictions, reformulation techniques are commonly adopted to transform a general DCOP into one where each (pseudo-)agent controls exclusively one variable [1,33]. This transformation can be inefficient in terms of agent computation and coordination, as it may limit the agents' ability to interact in pruning the search space [5]. While one can trivially extend existing algorithms to allow each agent to solve its local sub-problem (i.e., the value assignment of its local variables) in a centralized fashion, each sub-problem is still NP-hard, and can require a large amount of time if solved naively. This concern is true especially for application domains where agents may control a large number of local variables with large numbers of local constraints. We explore meeting scheduling problems as one such application domain in our experimental evaluations.

   In this paper, we introduce a general framework, called *Distributed MCMC* (DMCMC), which is based on a *Dynamic Programming*-based DCOP procedure [21]; the framework allows each agent to solve its local sub-problem using *Markov Chain Monte Carlo* (MCMC) sampling algorithms and uses general-purpose *Graphical Processing Units* (GPUs) to parallelize and speed up this process. We demonstrate the generality of this framework using two popular MCMC algorithms, the Gibbs [6] and Metropolis-Hastings [8,17] algorithms. Our experiments show that our framework is able to find better solutions up to two orders of magnitude faster than MGM and MGM2 (two incomplete DCOP algorithms). Additionally, it finds solutions that are within a 5% error of the optimum for problems that can be solved optimally. While the description of our solution focuses on DCOPs, our approach is also suitable to solve *Weighted Constraint Satisfaction Problems* (WCSPs).

2   Background

WCSPs: A *Weighted Constraint Satisfaction Problem* (WCSP) [11,27] is a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{F} \rangle$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a finite set of variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite domains for the variables in $\mathcal{X}$, with D_i being the set of possible values for the variable x_i, $\mathcal{F}$ is a set of *weighted constraints* (or *utility functions*). A weighted constraint $f_i \in \mathcal{F}$ is a function, $f_i : \times_{x_j \in \mathbf{x}^{f_i}} D_i \to \mathbb{R}^+ \cup \{-\infty\}$, where $\mathbf{x}^{f_i} \subseteq \mathcal{X}$ is the set of variables relevant to f_i, referred to as the *scope* of f_i. A *solution* σ is a value assignment to a set of variables $X_\sigma \subseteq \mathcal{X}$ that is consistent with the variables' domains. The utility $\mathcal{U}(\sigma) = \sum_{f \in \mathcal{F}, \mathbf{x}^f \subseteq X_\sigma} f(\sigma)$ is the sum of the utilities of all the applicable utility functions in σ. A solution is said *complete* if $X_\sigma = \mathcal{X}$. The goal is to find an optimal complete solution $\sigma^* = \operatorname{argmax}_\sigma \mathcal{U}(\sigma)$.

DCOPs: When the elements of a WCSP are distributed among a set of autonomous agents, we refer to it as a *Distributed Constraint Optimization Problem* (DCOP) [18,21,32]. Formally, a DCOP is described by a tuple

$\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$, where $\mathcal{X}$, $\mathcal{D}$ and $\mathcal{F}$ are the set of variables, their domains, and the set of utility functions, defined as in a classical WCSP, $\mathcal{A} = \{a_1, \ldots, a_m\}$ $(m \leq n)$ is a set of autonomous agents, and $\alpha : \mathcal{X} \to \mathcal{A}$ is a surjective function, from variables to agents, which assigns the control of each variable $x \in \mathcal{X}$ to an agent $\alpha(x)$. The goal in a DCOP is to find a complete solution that maximizes its utility: $\sigma^* = \mathrm{argmax}_\sigma \, \mathcal{U}(\sigma)$.

Given a DCOP P, $G = (\mathcal{A}, E)$ is the *constraint graph* of P, where $(i, j) \in E$ iff $\exists f \in \mathcal{F}$, where $\exists x_i, x_j \in \mathcal{X}$ with $\alpha(x_i) = a_i$ and $\alpha(x_j) = a_j$ s.t. $\{x_i, x_j\} \subseteq \mathbf{x}^f$. A *DFS pseudo-tree* arrangement for G is a *spanning tree* $T = \langle \mathcal{A}, E_T \rangle$ of G s.t. if $f \in \mathcal{F}$ and $\exists x_i, x_j \in \mathcal{X}$ with $\alpha(x_i) = a_i$ and $\alpha(x_j) = a_j$ s.t. $\{x_i, x_j\} \subseteq \mathbf{x}^f$, then x and y appear in the same
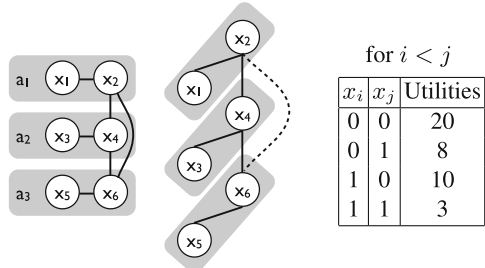
(a) Constraint Graph (b) Pseudo-tree (c) Constraint Table

Fig. 1. Example DCOP

branch of T. Edges of G that are *in* (resp. *out* of) E_T are called *tree edges* (resp. *backedges*). Tree edges connect a node with its parent and its children, while backedges connect a node with its *pseudo-parents* and its *pseudo-children*. We write $a_i \succ_T a_j$ if agent a_i is an ancestor of a_j in the pseudo-tree T. We use C_i, P_i, and $sep(a_i)$ to refer to, respectively, the set of child agents, the parent agent, and to the *separator* of agent a_i in the pseudo-tree. The latter is the set of variables owned by the a_i's ancestor agents that are constrained with variables owned by a_i or by its descendant agents.

Definition 1. *For each agent $a_i \in \mathcal{A}$, $L_i = \{x_j \in \mathcal{X} \mid \alpha(x_j) = a_i\}$ is the set of its* local *variables. $B_i = \{x_j \in L_i \mid \exists x_k \in \mathcal{X} \wedge \exists f_s \in \mathcal{F} : \alpha(x_k) \neq a_i \wedge \{x_j, x_k\} \subseteq \mathbf{x}^{f_s}\}$ is the set of its* interface *variables.*

Definition 2. *For each agent $a_i \in \mathcal{A}$, its* local constraint graph $G_i = (L_i, E_{\mathcal{F}_i})$ *is a subgraph of the constraint graph, where $\mathcal{F}_i = \{f_j \in \mathcal{F} \mid \mathbf{x}^{f_j} \subseteq L_i\}$.*

Figure 1(a) shows the constraint graph of a sample DCOP with 3 agents a_1, a_2, and a_3, where $L_1 = \{x_1, x_2\}$, $L_2 = \{x_3, x_4\}$, $L_3 = \{x_5, x_6\}$, $B_1 = \{x_2\}$, $B_2 = \{x_4\}$, and $B_3 = \{x_6\}$. The domains are $D_1 = \cdots = D_6 = \{0, 1\}$. Figure 1(b) shows one possible pseudo-tree (the dotted line is a *backedge*). Figure 1(c) shows the constraints.

DPOP: *Distributed Pseudo-tree Optimization Procedure* (DPOP) [21] is a complete DCOP algorithm. composed of three phases:[1]

[Phase 1] Pseudo-tree Generation: DPOP agents constructs a pseudo-tree using existing distributed pseudo-tree construction methods [7].

---

[1] It is a distributed variant of Bucket Elimination [2].

x_2	x_4	Utilities
0	0	max(60,24,50,19) = 60
0	1	max(50,19,40,14) = 50
1	0	max(50,19,40,14) = 50
1	1	max(40,14,30, 9) = 40

(a) Computations of a_3

x_2	Utilities
0	max(100,66,90,61) = 100
1	max(80,51,70,46) = 80

(b) Computations of a_2

Utilities
max(120,110) = 120

(c) Computations of a_1
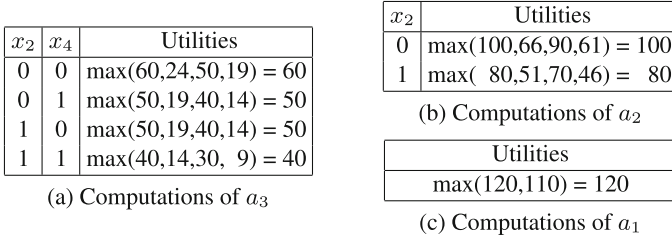
Fig. 2. Example UTIL phase computations

Algorithm 1. METROPOLIS-HASTING($\mathbf{z}$)

---

1 $\mathbf{z}^{(0)} \leftarrow$ INITIALIZE($\mathbf{z}$)
2 **for** $t = 1$ **to** T **do**
3 $\quad \mathbf{z}^* \leftarrow$ SAMPLE($q(\mathbf{z}^* \mid \mathbf{z}^{(t-1)})$)
4 $\quad \mathbf{z}^{(t)} \leftarrow \begin{cases} \mathbf{z}^* \text{ with } p = \min(1, \frac{\tilde{\pi}(\mathbf{z}^*)q(\mathbf{z}^{(t-1)},\mathbf{z}^*)}{\tilde{\pi}(\mathbf{z}^{(t-1)})q(\mathbf{z}^*,\mathbf{z}^{(t-1)})}) \\ \mathbf{z}^{(t-1)} \text{ with } 1-p \end{cases}$

---

5 **for** $i = 1$ **to** n **do**
6 $\quad z_i^t \leftarrow$ SAMPLE($\frac{1}{Z_\pi} \tilde{\pi}(z_i \mid z_1^t, \ldots, z_{i-1}^t, z_{i+1}^{t-1}, \ldots, z_n^{t-1})$)

[Phase 2] Utility Propagation: Each agent, starting from the leafs of the pseudo-tree, computes the optimal sum of utilities in its subtree for each value combination of variables in its separator. The agent does so by summing the utilities of its constraints with the variables in its separator and the utilities in the UTIL messages received from its children agents, and then projecting out its own variables by optimizing over them. In our example problem, agent a_3 computes the optimal utility for each value combination of variables x_2 and x_4 (see Fig. 2(a)), and sends the utilities to its parent agent a_2 in a UTIL message. Agent a_2 then computes the optimal utility for each value of the variable x_2 (see Fig. 2(b)), and sends the utilities to its parent agent a_1 in a UTIL message. Finally, agent a_1 computes the optimal utility of the entire problem (see Fig. 2(c)).

[Phase 3] Value Propagation: Each agent, starting from the root of the pseudo-tree, determines the optimal value for its variables. The root agent does so by choosing the values of its variables from its UTIL computation. In our example, agent a_1 determines that the values for both its variables leading to the largest utility are both 0 (with a overall utility of 120). It then sends the value of variable x_2 to its child agent a_2 in a VALUE message. Upon receiving the VALUE message from its parent agent, agent a_2 determines that the value with the largest utility for both its variables, assuming that $x_2 = 0$, is 0, with a utility of 100. In turn, it sends the value of variables x_2 and x_4 to its child agent a_3 in another VALUE message. Finally, upon receiving the VALUE message from its parent agent, agent a_3 determines that the value with the largest utility for both its variables, assuming that $x_2 = 0$ and $x_4 = 0$, is 0, with a utility of 60.

MCMC Sampling Algorithms: *Markov Chain Monte Carlo* (MCMC) sampling algorithms are commonly used to solve the *Maximum A Posteriori* (MAP) estimation problem. Recently, Nguyen *et al.* [19] have shown that DCOPs can be mapped to MAP estimation problems, allowing the use of MCMC algorithms to solve DCOPs. However, this mapping assumes that the constraint utilities are bounded, as they are normalized into distribution functions that MCMC algorithms aim to approximate. Therefore, MCMC algorithms cannot be used to solve DCOPs with hard constraints. Let us describe two popular MCMC algorithm—Gibbs [6] and Metropolis-Hastings [8,17].

Suppose we have a joint probability distribution $\pi(\mathbf{z})$ over n variables, $\mathbf{z} = z_1, z_2, \ldots, z_n$, that we would like to approximate. Moreover, suppose that it is easy to evaluate $\pi(\mathbf{z})$ for any given $\mathbf{z}$ up to some normalizing constant Z_π, such that: $\pi(\mathbf{z}) = \frac{1}{Z_\pi}\tilde{\pi}(\mathbf{z})$, where $\tilde{\pi}(\mathbf{z})$ can be easily computed but Z_π may be unknown. In order to draw the samples $\mathbf{z}$ to be fed to $\tilde{\pi}(\cdot)$, we use a *proposal distribution* $q(\mathbf{z}|\mathbf{z}^{(\tau)})$, from which we can easily generate samples, each depending on the current state $\mathbf{z}^{(\tau)}$ of the process. The latter can be interpreted as saying that when the process is in the state $\mathbf{z}^{(\tau)}$, we can generate a new state $\mathbf{z}$ from $q(\mathbf{z} \mid \mathbf{z}^{(\tau)})$. The proposal distribution is thus used to generate a sequence of samples $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots$, which forms a Markov chain.

Algorithm 1 shows the pseudocode of the *Metropolis-Hastings* algorithm. It first initializes $\mathbf{z}^{(0)}$ to any arbitrary value of the variables $z_1, \ldots, z_n$ (Line 1). Then, it iteratively generates a candidate $\mathbf{z}^*$ for $\mathbf{z}^{(t)}$ by sampling from the proposal distribution $q(\mathbf{z}^* \mid \mathbf{z}^{(t-1)})$ (Line 3). The candidate sample is then accepted with probability p (Line 4). If the sample is accepted, then $\mathbf{z}^{(t)} = \mathbf{z}^*$, otherwise $\mathbf{z}^{(t-1)}$ is left unchanged. This process continues for a fixed number of iterations or until convergence [25] is achieved.

The *Gibbs* sampling algorithm is a special case of the Metropolis-Hastings algorithm, where Line 3 is replaced by Lines 5–6. Additionally, note that Gibbs requires the computation of the normalizing constant Z_π while Metropolis-Hasting does not, as the calculation of the proposal distribution does not require that information. This is desirable when the computation of the normalizing constant becomes prohibitive (e.g., with increasing problem dimensionality). In this paper, we describe how one could parallelize the operations of MCMC sampling algorithms using GPU hardware.

GPUs: Modern *Graphics Processing Units* (GPUs) are multiprocessor devices, offering thousands of computing cores to support graphical processing. In this paper, we use the *Compute Unified Device Architecture* (CUDA) programming model proposed by NVIDIA [26], which enables the use of the multiple cores of a graphic card to accelerate general (non-graphical) applications by providing programming models and APIs that enable the full programmability of the GPU. The underlying model of parallelism supported by CUDA is *Single-Instruction Multiple-Thread* (SIMT), where the same instruction is executed by different threads that run on identical cores, while data and operands may differ from thread to thread.

A typical CUDA program is a C/C++ program that includes parts meant for execution on the CPU (referred to as the *host*) and parts meant for parallel execution on the GPU (referred as the *device*). A parallel computation is described by a collection of *kernels*, where each kernel is a function to be executed by several threads. To facilitate the mapping of the threads to the data structures being processed, threads are grouped in *blocks*, and have access to several memory levels, each with different properties in terms of speed, organization (e.g., multiple banks that can be concurrently accessed), and capacity. Each thread stores its private variables in very fast registers. Threads within a block can communicate by reading and writing a common area of memory (called *shared memory*). Communication between blocks and communication between blocks and the host (i.e., the CPU) is realized through a large (but slow) global memory.

3   Distributed MCMC Framework

We now describe our *Distributed MCMC* (DMCMC) framework, which extends centralized MCMC sampling algorithms and DPOP. At a high level, its operations are similar to those of DPOP, except that the computation of the utility tables sent by agents during the UTIL phase is done by sampling with GPUs. Notice that the computation of each row in a utility table is independent of the computations in the other rows. Thus, DMCMC exploits this independence and samples the values in each row in parallel.

Algorithm 2 shows the pseudocode of DMCMC for an agent a_i. It takes as inputs R, the number of sampling runs to perform from different initial value assignments, and S, the number of sampling trials. Like DPOP, DMCMC also exhibits three phases. The first phase is identical to that of DPOP (Line 7). In the second phase:

- Each agent a_i calls GPU-INITIALIZE() to set up the GPU kernel specifics (e.g., number of threads and amount of shared memory to be assigned to each block, and to initialize the data structures on the GPU device memory) (Line 8). The GPU kernel settings are decided according to the shared memory requirements and the number of registers used by the successive function call, in order to maximize the number of blocks that can run in parallel—this step can be automated.
- Each agent a_i, *in parallel*, calls GPU-MCMC-SAMPLE() which performs the local MCMC sampling process to compute the best utility and the corresponding solution (value assignments for all non-interface local variables $x_i^j \in L_i \setminus B_i$) for each combination of values of the interface variables $x_i^k \in B_i$ (Line 9). This computation process is done via sampling with GPUs and the results are then transferred from the device to the host (Line 10). In our example in Fig. 1, agent a_3 determines that its best utility is 20 if its interface variable $x_6 = 0$, and 8 if $x_6 = 1$. This utility table is stored in $UTIL_{a_i}$. Note that all the agents call this procedure immediately after the pseudo-tree is constructed. In

Algorithm 2. DMCMC(R, S)

---

7 Generate pseudo-tree
8 GPU-INITIALIZE( )
9 $\langle M_i^1, U_i^1 \rangle, \ldots, \langle M_i^R, U_i^R \rangle \leftarrow$ GPU-MCMC-SAMPLE(R, S)
10 $UTIL_{a_i} \leftarrow$ GET-BEST-SAMPLE($\langle M_i^1, U_i^1 \rangle, \ldots, \langle M_i^R, U_i^R \rangle$)
11 **if** $C_i = \emptyset$ **then**
12 $UTIL_{a_i} \leftarrow$ CALCUTILS( )
13   Send $UTIL$ message ($a_i, UTIL_{a_i}$) to P_i
14 Activate UTILMessageHandler($\cdot$)
15 Activate VALUEMessageHandler($\cdot$)

Procedure VALUEMessageHandler($a_k, VALUE_{a_k}$)

---

16 $VALUE_{a_i} \leftarrow VALUE_{a_k}$
17 **for** $x_i^j \in L_i$ **do** $d_i^{j*} \leftarrow$ CHOOSEBESTVALUE($VALUE_{a_i}$) ;
18 **for** $a_c \in C_i$ **do**
19 $VALUE_{a_i} \leftarrow \{(x_i^j, d_i^{j*}) \,|\, x_i^j \in sep(a_c)\} \cup \{(x_k, d_k^*) \in VALUE_{a_k} \,|\, x_k \in sep(a_c)\}$
20   Send $VALUE$ message ($a_i, VALUE_{a_i}$) to a_c

---

contrast, agents in DPOP compute the best utility only after receiving UTIL messages from all children agents.

- Each agent a_i computes the utilities for the constraints between its interface variables and variables in its separator, joins them with the sampled utilities (Line 12), and sends them to its parent (Line 13). The agent repeats this process each time it receives a UTIL message from a child (Lines 20–27).

At the end of the second phase (Line 23), like in DPOP, the root agent will know the overall utility for each combination of values of its variables $x_i^j \in B_i$. It chooses its best value combination that results in the maximum utility (Line 25), and starts the third phase by sending to each child agent a_c the values of variables $x_i^j \in sep(a_c)$ that are in the separator of the child (Lines 26–28). The *MessageHandlers* of Lines 14 and 15 are activated for any new incoming message.

3.1 GPU Data Structures

In order to fully capitalize on the parallel computational power of GPUs, the data structures need to be designed in such a way to limit the amount of information exchanged between the CPU host and the GPU devices. Each DMCMC agent stores all the information it needs in the GPU global memory. This allows each agent running on a GPU device to communicate with the CPU host only once, which is at the end of the sampling process, to transfer the results. Each agent a_i maintains the following information:

- Its *local* variables $L_i \subseteq \mathcal{X}$ (including its interface variables $B_i \subseteq L_i$).
- The domains of its local variables, D_i (assumed to have all equal size for simplicity).

Procedure UTILMessageHandler(a_k, $UTIL_{a_k}$)

---

21 Store $UTIL_{a_k}$
22 **if** received $UTIL$ message from each child $a_c \in C_i$ **then**
23 $UTIL_{a_i} \leftarrow$ CALCUTILS( )
24   **if** $P_i = NULL$ **then**
25   **for** $x_i^j \in L_i$ **do** $d_i^{j*} \leftarrow$ CHOOSEBESTVALUE($\emptyset$) ;
26   **for** $a_c \in C_i$ **do**
27 $VALUE_{a_i} \leftarrow \{(x_i^j, d_i^{j*}) \mid x_i^j \in sep(a_c)\}$
28   Send $VALUE$ message (a_i, $VALUE_{a_i}$) to a_c
29   **else** Send $UTIL$ message (a_i, $UTIL_{a_i}$) to P_i ;

Function CalcUtils( )

---

30 $UTIL_{sep} \leftarrow$ utilities for all value comb. of $x_i \in B_i \cup sep(a_i)$
31 $UTIL_{a_i} \leftarrow$ JOIN($UTIL_{a_i}$, $UTIL_{sep}$, $UTIL_{a_c}$) for all $a_c \in C_i$
32 $UTIL_{a_i} \leftarrow$ PROJECT(a_i, $UTIL_{a_i}$)
33 **return** $UTIL_{a_i}$

- A matrix M_i of size $|D_i|^{|B_i|} \times |L_i|$, where the j-th row is associated with the j-th permutation of the interface variable values, in lexicographic order, and the k-th column is associated with the k-th variable in L_i. The matrix columns associated with the local variables in L_i are initialized with random value assignments in $[0, |D_i|-1]$. At the end of the sampling process it contains the converged domain values of the local variables for each value combination of the interface variables.
- A vector U_i of size $|D_i|^{|B_i|}$, which stores the utilities of the solutions in M_i.
- The *local constraint graph* G_i, which includes the constraints in $\mathcal{F}_i$.

The GPU-INITIALIZE() procedure of Line 8 stores the data structures above for each agent on its CUDA device. All the data stored on the GPU devices is organized in mono-dimensional arrays, so as to facilitate *coalesced memory accesses*. The set of local variables L_i are ordered, for convenience, in lexicographic order and so that the interface variables B_i are listed first.

3.2   Local Sampling Process

The GPU-MCMC-SAMPLE procedure of Line 9 is the core of the local sampling algorithm, and can be performed by any MCMC sampling method. It executes S sampling trials for the subset of non-interface local variables $L_i \setminus B_i$ of agent a_i. Since the MCMC sampling procedure is stochastic, we can run R parallel sampling processes with different initial value assignments and take the best utility and corresponding solution across all runs. Each parallel run is executed by a *group of CUDA blocks*. Independent operations within each sample are also exploited in parallel using *groups of threads* within each block. For example, the proposal distribution adopted by Gibbs is computed using $|D_i|$ parallel *threads*.

Procedure GPU-MCMC-Sample(R, S)

---

34 $\langle \mathbf{z}, \mathbf{z}^*, [q, Z_\pi], G_i \rangle \leftarrow \text{AssignSharedMem}()$

35 $r_{id} \leftarrow$ the thread's row index of M_i

36 $\mathbf{z} \overset{|L_i|}{\Longleftarrow} M_i[r_{id}]$

37 $\langle \mathbf{z}^*, util^* \rangle \leftarrow \langle \mathbf{z}, \sum_{f_j \in \mathcal{F}_i} f_j(\mathbf{z}_{|\mathbf{x}^{f_j}}) \rangle$

38 **for** $t = 1$ **to** S **do**

39 $\quad \mathbf{z} \overset{k}{\Longleftarrow} \text{Sample}(q(\mathbf{z} \mid \mathbf{z}^{(t-1)}))$ w/ prob. $\min\{1, \frac{\tilde{\pi}(\mathbf{z})}{\tilde{\pi}(\mathbf{z}^{(t-1)})}\}$

40 $\quad util \leftarrow \sum_{f_j \in \mathcal{F}_i} f_j(\mathbf{z}_{|\mathbf{x}^{f_j}})$

41 $\quad$ **if** $util > util^*$ **then** $\langle \mathbf{z}^*, util^* \rangle \leftarrow \langle \mathbf{z}, util \rangle$;

42 $\langle M_i^R[r_{id}], U_i^R[r_{id}] \rangle \leftarrow \langle \mathbf{z}^*, util^* \rangle$

Figure 3 illustrates the different parallelizations performed by the GPU-MCMC-Sample process with Gibbs.

The general GPU-MCMC-Sample procedure is shown in Lines 34–42 and we use the symbols $\leftarrow$ and $\overset{k}{\Longleftarrow}$ to denote sequential (single thread) and parallel (k threads) operations, respectively. We also denote with n the size of the state $\mathbf{z}$ being sampled, with $n = |L_i| - |B_i|$. The function takes in as inputs the number of desired sampling trials S and the number of parallel sampling runs R. It first assigns the shared

Fig. 3. Parallelization illustration

memory allocated to the arrays $\mathbf{z}$ and $\mathbf{z}^*$, which are used to store the current and best sample of value assignments for all local variables, respectively; the local constraint graph G_i; and, if the MCMC sampling algorithm requires computing the normalization constant of the proposal distribution explicitly, the array q and Z_π, which are used to store the probabilities for each value of the non-interface local variables and the normalization constant, respectively (Line 34).

Each thread identifies its row index r_{id} of the matrix M_i, initializes its sample with the values stored in $M_i[r_{id}]$, calculates the utility for that sample, and stores the initial sample and utility as the best sample and utility found so far (Lines 35–37). It then runs S sampling trials, where in each trial, it samples a new state $\mathbf{z}$ from a proposal distribution $q(\mathbf{z} \mid \mathbf{z}^{(t-1)})$ and updates that state according to the accept/reject probabilities described in the MCMC background (Line 39).

The proposal distribution q and the accept/reject probabilities depend on the choice of MCMC algorithm. We now describe them for Metropolis-Hasting and Gibbs.

- **Metropolis-Hastings:** The proposal distribution that we adopt is a multivariate normal distribution $q \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with $\boldsymbol{\mu}$ being a n-dimensional vector

---

Procedure CUDA Gibbs Proposal Distribution Calculation

---

43 $d_{id} \leftarrow$ the thread's value index of D_i

44 **for** $k = |B_i|$ **to** $|L_i| - 1$ **do**

45 $\quad q[d_{id}] \overset{|D_i|}{\Leftarrow} \exp \left[\sum_{f_j \in \mathcal{F}_i} f_j(\mathbf{z}_{|\mathbf{x}^{f_j}}) \right]$

46 $\quad Z_\pi \leftarrow \sum_{i=0}^{|D_i|-1} q[i]$

47 $\quad q[d_{id}] \overset{|D_i|}{\Leftarrow} q[d_{id}] \cdot \frac{1}{Z_\pi}$

48 $\quad \mathbf{z} \leftarrow \text{SAMPLE}(q(\mathbf{z} \mid \mathbf{z}^{(t-1)}))$

---

of mean values, whose elements $\mu_j^{(t)}$ have the value of the corresponding component in the previous sample $z_j^{(t-1)}$ and $\boldsymbol{\Sigma}$ is the covariance matrix defined with the only non-zero elements being their diagonal ones and set to be all equal to $\sqrt{D_i}$. We compute the proposal distribution q using n parallel threads. The proposal distribution for Metropolis-Hastings is symmetric and, thus, the accept/reject probabilities are simplified as shown in Line 39.

- **Gibbs:** For Gibbs, Line 39 needs to be replaced with Lines 43–48. Gibbs sequentially iterates through all the non-interface local variable $x_k \in L_i \setminus B_i$ and computes in parallel the probability $q[d_{id}]$ of each value d_{id} according to the equation:

$$q(x_k = d_{id} \mid x_l \in L_i \setminus \{x_k\}) = \frac{1}{Z_\pi} \exp \sum_{f_j \in \mathcal{F}_i} f_j(\mathbf{z}_{|\mathbf{x}^{f_j}})$$

where $\mathbf{z}_{|\mathbf{x}^{f_j}}$ is the set of value assignments for the variables in the scope of constraint f_j, and Z_π is the normalizing constant. We compute q using $|D_i|$ parallel threads.

To ensure that the procedure returns the best sample found, we verify whether there is an improvement on the best utility (Lines 40–41). At the end of the sampling trials, it stores its best sample and utility in the r_{id}-th row in the matrix M_i and vector U_i, respectively (Line 42).

4   Theoretical Properties

We now relate the quality of DCOP solutions to MCMC sampling strategies, and provide some complexity analyses of the DMCMC algorithms.

Let us first introduce some background on Markov Chains and on the structural properties that they need to satisfy to guarantee convergence to a stationary distribution.

Let $\mathbf{Z} = (\mathbf{z}^0, \mathbf{z}^1, \ldots, \mathbf{z}^t, \ldots)$, with $\mathbf{z}^t \in \mathbf{D} \subseteq \mathbb{R}$ be a *Markov chain* with finite state space $\mathbf{S} = \{s_1, s_2, \ldots, s_L\}$ and a $L \times L$ *transition matrix* T whose entries define the probability of transitioning from one state to another as $P(\mathbf{z}^{t+1} = s_j \mid \mathbf{z}^t = s_i) = T_{ij}$.

The Markov chain $\mathbf{Z}$ converges to a stationary distribution if it is *irreducible* and *aperiodic*. These two concepts are introduced as follows.

Definition 3 (Irreducibility). *A Markov chain is* irreducible *if it is possible to reach any state from any other state using only transitions of positive probability. That is,* $\forall s_i, s_j \in \mathbf{S}, \exists m < \infty : P(\mathbf{z}^{t+m} = s_j \,|\, \mathbf{z}^t = s_i) > 0$ *for a given instance t.*

Definition 4 (Periodicy). *A state* $s_i \in \mathbf{S}$ *has a period k if any return of the chain in it is possible with multiple of k time steps. The period of a state is defined as* $k = gcd\{t : P(\mathbf{z}^t = s_i \mid \mathbf{z}^0 = s_i) > 0\}$*, where gcd is the greatest common divisor. A state is said to be aperiodic if* $k = 1$*, that is, visits of the Markov chain to such state (i.e.,* $P(\mathbf{z}^t = s_i \mid \mathbf{z}^0 = s_i) > 0$*) can occur at irregular times. A Markov chain is said to be* aperiodic *if every state in* $\mathbf{S}$ *is aperiodic.*

Note that for an irreducible Markov chain, if at least one state is aperiodic, then the whole Markov chain is aperiodic.

We now provide bounds on convergence rates for the DMCMC algorithms based on MCMC sampling.

Definition 5 (Top α_i-Percentile Solutions). *For an agent* a_i *the top* α_i*-percentile solutions* S_{α_i} *is a set containing solutions for the local variables* L_i *that are no worse than any solution in the supplementary set* $D_i \setminus S_{\alpha_i}$*, and* $\frac{|S_{\alpha_i}|}{|D_i|} = \alpha_i$*. Given a list of agents* $a_1, \ldots, a_m$*, the top* $\bar{\alpha}$*-percentile solutions* $S_{\bar{\alpha}}$ *is defined as* $S_{\bar{\alpha}} = S_{\alpha_1} \times \ldots \times S_{\alpha_m}$*.*

Property 1. After $N_i = \frac{1}{\alpha_i \epsilon_i}$ samples with an MCMC algorithm T, the probability that the best solution found thus far $\mathbf{z}_{N_i}$ is in the top α_i for an agent a_i is at least $1 - \epsilon_i$:

$$P_{\mathrm{T}}\left(\mathbf{z}_{N_i} \in S_{\alpha_i} \mid N_i = \frac{1}{\alpha_i \cdot \epsilon_i}\right) \geq 1 - \epsilon_i.$$

Definition 5 and Property 1 are introduced by Nguyen *et al.* [19] and can be generalized to any MCMC sampling algorithm whose Markov chain generated is irreducible and aperiodic as convergence is guaranteed in a finite number of time steps.

Theorem 1. *Given m agents* $a_1, \ldots, a_m \in \mathcal{A}$*, and a number of samples* $N_i = \frac{1}{\alpha_i \cdot \epsilon_i}$ *($i = 1, \ldots, m$), the probability that the best complete solution found thus far* $\mathbf{z}_{\mathbf{N}}$ *is in the top* $\bar{\alpha}$*-percentile is greater than or equal to* $\prod_{i=1}^m (1 - \epsilon_i)$*, where* $\mathbf{N} = \bigwedge_{i=1}^m N_i$*. In other words,*

$$P_{\mathrm{T}}\left(\mathbf{z}_{\mathbf{N}} \in S_{\bar{\alpha}} \mid \mathbf{N}\right) \geq \prod_{i=1}^m (1 - \epsilon_i).$$

Proof. Let $\mathbf{z}_N$ denote the best solution found so far in the process resolution and $\mathbf{z}_{N_i}$ denote the best partial assignment over the variables held by agent a_i found after N_i samples. Let $\mathbf{S}_i$ be a random variable describing whether $\mathbf{z}_{N_i} \in S_{\alpha_i}$. Thus:

$$P_T(\mathbf{z_N} \in S_{\bar{\alpha}} \mid \mathbf{N}) \tag{1a}$$

$$= P_T(\mathbf{z_N} \in S_{\bar{\alpha}} \mid \mathbf{N}_1, \ldots, \mathbf{N}_m) \tag{1b}$$

$$= P_T(\mathbf{z_N} \in S_{\alpha_1} \times \ldots \times S_{\alpha_m} \mid \mathbf{N}_1, \ldots, \mathbf{N}_m) \tag{1c}$$

$$= P_T(\mathbf{S}_1, \ldots, \mathbf{S}_m \mid \mathbf{B}_1, \ldots, \mathbf{B}_m, \mathbf{N}_1, \ldots, \mathbf{N}_m) \tag{1d}$$

where each $\mathbf{B}_i$ $(i = 1, \ldots, m)$ is a random variable describing a particular value assignment associated to the interface variables B_i for the agent a_i. They are introduced to relate each of the $\mathbf{z_{N_i}}$ to each other, which are sampled independently.

Since the values sampled in the local variable of a_i are dependent only of the values of the interface values B_i, it follows that $\mathbf{S}_i$ is conditionally dependent of $\mathbf{B}_i$ but conditionally independent of all other $\mathbf{B}_j$, with $j \neq i$:

$$S_i \perp\!\!\!\perp B_j \mid B_i$$

for all $j = 1 \ldots m$ and $j \neq i$. Noticing that, given random variables a, b, c, whenever $a \perp\!\!\!\perp b \mid c$ we can write: $P(a \mid b, c) = P(a \mid c)$, and that $P(a, b \mid c) = P(a \mid b, c)$, it follows that Eq. (1d) can be rewritten as:

$$P_T(\mathbf{S}_1 \mid \mathbf{B}_1, \mathbf{N}_1) \cdot \ldots \cdot P_T(\mathbf{S}_m \mid \mathbf{B}_m, \mathbf{N}_m)$$

$$= P_T(\mathbf{z_{N_1}} \in S_{\alpha_1} \mid \mathbf{B}, \mathbf{N}) \cdot \ldots \cdot P_T(\mathbf{z_{N_m}} \in S_{\alpha_m} \mid \mathbf{B}, \mathbf{N}) \tag{2a}$$

$$\geq (1 - \epsilon_1) \cdot \ldots \cdot (1 - \epsilon_m) \tag{2b}$$

$$= \prod_{i=1}^{m} (1 - \epsilon_i). \tag{2c}$$

for any of the assignments of the variables in B_i, as the utility functions involving variables in the interface of any two agents are solved optimally. $\qquad\square$

Theorem 2. *The number of messages required by DMCMC is $O(|\mathcal{A}|)$.*

Proof. DMCMC agents exchange $|\mathcal{A}| - 1$ UTIL messages (one through each tree-edge) and $|\mathcal{A}| - 1$ VALUE messages. Thus, the total number of messages required by the algorithm is $O(|\mathcal{A}|)$. $\qquad\square$

Note that, unlike DPOP, which requires $O(|\mathcal{X}|)$ messages, no message exchange is required to solve the constraints defined over the scope of the local variables each agent, which is achieved via local sampling.

Theorem 3. *The memory complexity of each DMCMC agent $a_i \in \mathcal{A}$ is $O(|D_i|^{|S_i \setminus B_i|})$, where $S_i = \{x | x \in sep(a_i) \wedge \alpha(x) \succ_T a_i \wedge \exists f \in \mathcal{F}.x \in \mathbf{x}^f \wedge \mathbf{x}^f \cap B_i \neq \emptyset\}$, is the set of the ancestors agent's variables in its separator which are involved in a constraint with some variable in B_i.*

Proof. Each agent $a_i \in \mathcal{A}$ needs to store its own utilities and the corresponding solution (value assignment for all non-interface local variables $x_i^j \in L_i \setminus B_i$) for each combination of values of the interface variables $x_i^k \in B_i$, thus requiring

$O(|D_i|^{|B_i|})$ space. Moreover during the UTIL propagation phase, each agent a_i stores the UTIL messages of each of its children $a_c \in C_i$, which also sends messages of size $O(|D_i|^{|S_c \setminus B_c|})$. Joint and projection operations can be performed efficiently within $O(|D_i|^{|S_i \setminus B_i|})$ space. Thus the memory complexity of each agent is exponential in its *induced width*, $O(|D_i|^{|S_i \setminus B_i|})$. $\square$

One can bound the maximum message size and serialize large messages by letting the backedge handlers ask explicitly for solutions and utilities for a subset of their values sequentially. Moreover, one could reduce the memory requirements at cost of sacrificing completeness, by propagating solutions for a bounded set of value combinations rather than all combination of values of the interface variables. Several approaches have been proposed to reduce the memory requirement of DPOP [3,22,23].

5   Related Work

To the best of our knowledge, there are only two sampling algorithms developed to solve DCOPs thus far, namely DUCT [20] and Distributed Gibbs [19]. DUCT is a distributed version of the UCT algorithm [9]. It maintains and uses upper confidence bounds on each value of a variable to determine which value to choose during the sampling process. It updates the bounds to make them more informed after each sampling trial.

Like DMCMC (with Gibbs as the MCMC algorithm), Distributed Gibbs is also a distributed version of Gibbs. However, Distributed Gibbs uses different communication protocols and computation procedures, which results in slow convergence due to high network load requirements [19]. More specifically, Distributed Gibbs performs the Gibbs sampling process on the entire space of all variables (i.e., in each sampling trial, it assigns a value to each variable sequentially until all variables are assigned a value), while DMCMC performs multiple sampling processes in parallel, one for each subset of local variables of an agent. As a result, DMCMC is able to better exploit the parallel processes with the use of GPUs.

6   Experimental Results

We implemented CPU and GPU versions of the DMCMC framework with Gibbs (Gibbs) and Metropolis-Hastings (MH) as the MCMC sampling algorithms. The CPU versions sample sequentially, while the GPU versions sample in parallel with GPUs. We compare them against DPOP [21] (an optimal algorithm), MGM and MGM2 [15] (sub-optimal algorithms). We use publicly-available implementations of these algorithms, which are implemented in the FRODO framework [14]. We run our experiments on a Intel(R) Xeon(R) CPU, 2.4 GHz, 32 GB of RAM, Linux x86_64, equipped with a Tesla C2075, 14SM, 448-core, 1.15 clock rate, CUDA 2.0. Note that we do not parallelize at the level of CPU cores, thus the

number of cores in the CPU is immaterial. We measure the algorithms' runtime using the wall clock (*wct*) and the simulated time (*st*) [29] metrics, and perform evaluations on random graphs and meeting scheduling problems. All reported results are averaged among 100 runs. The underlying constraint graphs are generated as follows: We create an n-node network, whose local constraint graphs density p_1^ℓ produces $\lfloor |L_i|(|L_i| - 1)p_1^\ell \rfloor$ edges among the local variables of each agent a_i, and whose (global) density p_1^g produces $\lfloor b(b - 1)p_1^g \rfloor$ edges among non-local interface variables, where b is the total number of interface variables of the problem. All constraints utilities are randomly chosen from the interval $[1, 1000]$.

We first evaluate the effect of the initial parameters R and S for our DMCMC algorithms in a setting in which DPOP could terminate its execution, and thus report its (optimal) solution. We fix the number of agents to 5, the number of local variables for each agent to 10, their domain sizes to 10, and the graph densities $p_1^g = p_1^\ell = 0.5$. Figure 4(left) illustrates the runtime (in seconds) for the CPU and GPU implementations of our DMCMC algorithms for a range of the initial parameters $R \in [1, 100]$ and $S \in [10, 10000]$. These results shows that there is a clear benefit to parallelize the sampling operations with GPUs, exhibiting more than one order of magnitude speed up.

In the rest of the experiment, we show the GPU version only. Figure 4(right) reports the ratio of the quality of the solutions returned by Gibbs and MH at varying of the parameters S and R, over that returned by DPOP. Additionally, we report the average (solid line) and variance (dotted lines) solution quality returned by MGM2. We observe that the prediction quality increases with increasing R and T. Gibbs is slower than MH, as it requires the computation of the normalization constants, which are computationally expensive even when parallelized. However, Gibbs finds better solutions than MH. Additionally, Gibbs finds better solutions than MGM2 for $S > 20$, and MH finds solutions whose quality is comparable to those returned by MGM2.

Next, we evaluate our algorithms at varying of several problem parameters on meeting scheduling problems. In these problems, meetings need to be scheduled between members of a hierarchical organization, (e.g., employees of a company; students, faculty members, and staff of a university), taking restrictions in their availability as well as their priorities into account. We used the Private Events as Variables (PEAV) problem formulation, which is commonly used in the literature, where the variables model the meetings, their domains are the time slots when they can be held, and the constraints are between meetings that share participants [16]. In our experiments, we vary the number of agents $|\mathcal{A}| = \{5, 10, 25, 50\}$, the number of variables $|\mathcal{X}_i| = \{5, 10, 25, 50\}$ of each agent a_i, the domain size $|D_i| = \{12, 24, 48, 96\}$ of each variable x_i, the density of the local constraint graph $p_1^\ell = \{0.25, 0.5, 0.75, 1.0\}$ of each agent a_i. For each of the experiments below, we vary only one parameter and fix the rest in their "default" values: $|\mathcal{A}| = 10, |\mathcal{X}_i| = 10, |D_i| = 24, p_1^\ell = 0.5$. We set the number of samples for the D-MCMC algorithms to 100.
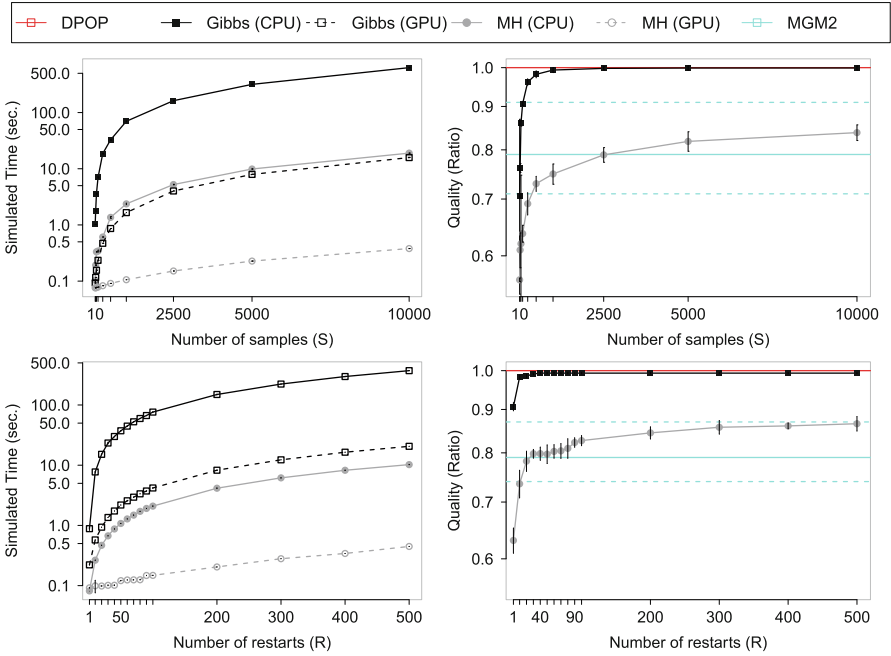
Fig. 4. Experimental results: random graph instances

Table 1 reports the runtime (in seconds) and solution qualities for all algorithms, where *oot* indicates that the algorithm timed out after 5 min of wall-clock time. The best runtimes and solution qualities are shown in bold. We make the following observations:

- In all parameter settings, DMCMC with Gibbs finds better solutions than MGM and MGM2. Additionally, while the runtime for Gibbs_{CPU} are comparable to those of MGM and MGM2, Gibbs_{GPU} found those solutions by one order of magnitude faster than MGM and MGM2.
- The solutions quality reported by DMCMC with MH are comparable to those reported by MGM and MGM2, and MH_{GPU} is at least one, and up to two order of magnitude times faster than MGM and MGM2.
- The GPU versions of our DMCMC algorithms are in general up to one order of magnitude faster than their CPU counterparts, and up to two orders when the local problem size increases. This result indicates that the GPUs can take advantage of the inherent parallelism present in the algorithm as a result of the partitioning of the problem into independent subproblems.
- Finally, for the problems for which DPOP successfully terminated within the time limit, we could measure the error in the quality of solutions found by DMCMC with Gibbs, which is only up to 5 %.

Due to the unavailability of a public implementation, we did not compare our approaches against DUCT, however, Nguyen *et al.* [19] showed that DPOP

Table 1. Experimental results: meeting scheduling problems

$|\mathcal{A}|$	5			10			25			50		
	wct	st	quality	wct	st	quality	wct	st	quality	wct	st	quality
DPOP	125.39	94.98	**1661**	oot	oot	-	oot	oot	-	oot	oot	-
MGM	7.435	0.435	1379	11.910	0.446	2766	24.211	0.417	6692	45.771	0.462	13802
MGM2	8.939	0.979	1389	23.903	1.526	2783	56.035	1.629	7116	112.54	1.788	14145
Gibbs$_{CPU}$	6.146	1.101	1638	12.093	1.190	3319	31.031	1.347	**8344**	62.411	1.489	**16577**
Gibbs$_{GPU}$	0.162	0.033	1635	0.301	0.034	**3338**	0.708	0.041	**8344**	1.416	0.048	16550
MH$_{CPU}$	0.561	0.113	1131	1.091	0.121	2775	2.281	0.176	6921	3.921	0.185	12112
MH$_{GPU}$	**0.047**	**0.014**	1143	**0.102**	**0.016**	2663	**0.196**	**0.017**	6925	**0.360**	**0.022**	11856
$|\mathcal{X}_i|$	5			10			25			50		
	wct	st	quality	wct	st	quality	wct	st	quality	wct	st	quality
DPOP	6.720	0.668	**1136**	oot	oot	-	oot	oot	-	oot	oot	-
MGM	5.260	0.242	947	11.910	0.446	2766	46.861	1.581	11652	180.05	5.749	35972
MGM2	8.701	0.602	941	23.903	1.526	2783	184.63	9.477	11889	oot	oot	-
Gibbs$_{CPU}$	2.336	0.489	1115	12.093	1.190	3319	182.68	2.446	13811	oot	oot	-
Gibbs$_{GPU}$	0.098	0.014	1104	0.301	0.34	**3338**	1.896	0.368	**13874**	12.707	1.384	**42124**
MH$_{CPU}$	0.351	0.048	986	1.091	0.121	2775	4.982	0.879	9850	56.077	6.506	33114
MH$_{GPU}$	**0.050**	**0.011**	972	**0.102**	**0.016**	2663	**0.146**	**0.022**	9716	**0.489**	**0.046**	32405
$|D_i|$	12			24			48			96		
	wct	st	quality	wct	st	quality	wct	st	quality	wct	st	quality
DPOP	22.230	9.996	**1332**	oot	oot	-	oot	oot	-	oot	oot	-
MGM	11.300	0.222	1077	11.910	0.446	2766	13.317	0.560	6133	18.177	1.106	13058
MGM2	19.723	0.541	1134	23.903	1.526	2783	53.972	5.314	6660	148.40	10.954	13866
Gibbs$_{CPU}$	3.348	0.530	1323	12.093	1.190	3319	51.669	5.716	7343	200.53	21.546	**16769**
Gibbs$_{GPU}$	0.214	0.029	1319	0.301	0.034	**3338**	0.763	0.090	**7357**	3.149	0.364	16278
MH$_{CPU}$	0.321	0.047	1086	1.091	0.321	2775	2.030	0.712	6135	6.081	1.306	12277
MH$_{GPU}$	**0.051**	**0.010**	1102	**0.102**	**0.016**	2663	**0.159**	**0.041**	6147	**0.218**	**0.146**	12189
p_1^ℓ	0.25			0.50			0.75			1.00		
	wct	st	quality	wct	st	quality	wct	st	quality	wct	st	quality
DPOP	10.850	0.885	**2305**	oot	oot	-	oot	oot	-	oot	oot	-
MGM	8.037	0.231	1835	11.910	0.446	2766	16.124	0.435	3342	19.832	0.605	3974
MGM2	12.908	0.708	1906	23.903	1.526	2783	47.258	2.554	3364	46.270	3.035	4091
Gibbs$_{CPU}$	7.991	0.981	2.269	12.093	1.190	3319	19.004	2.347	4032	25.691	2.821	**4751**
Gibbs$_{GPU}$	0.216	0.024	2300	0.301	0.034	**3338**	0.389	0.043	**4074**	0.451	**0.053**	4706
MH$_{CPU}$	0.775	0.101	1983	1.091	0.121	2775	1.225	0.135	3454	1.491	0.179	3921
MH$_{GPU}$	**0.090**	**0.013**	1931	**0.102**	**0.016**	2663	**0.170**	**0.021**	3458	**0.215**	**0.027**	3814

outperforms DUCT especially when the problem sizes are small. In contrast, our approach consistently outperforms DPOP even on small problems. Additionally, they show that Distributed Gibbs [19] requires a large number of iterations to converge since it is estimating the joint distribution of the entire problem. In contrast, our MCMC framework with Gibbs requires a much smaller number of iterations, since it is only estimating the joint distribution of agent's local variables.

7   Conclusions

Our work is motivated by several factors: (i) the assumption in most DCOP algorithms that each agents owns exactly one variable; (ii) the recent introduction of sampling-based DCOP algorithms, which have been shown to outperform existing incomplete DCOP algorithms; and (iii) the advances in GPUs. These combination of factors provides a unique opportunity for us to harness the power of parallel computation of GPUs to solve general DCOPs with multiple variables per agent. In this paper, we introduce the Distributed MCMC framework, which decomposes a DCOP into independent sub-problems that can each be sampled in parallel by GPUs. Our experimental results show that it can find good solutions up to one order of magnitude faster than MGM and MGM2. These results demonstrate the potential for using GPUs to scale up DCOP algorithms, which is exciting as GPUs provide access to thousands of computing cores at a very affordable cost. While the description of our solution focuses on DCOPs, our approach is also suitable to solve WCSPs. In the future, we plan to explore this direction, as well as extending the proposed framework to reduce its memory requirement similar to MB-DPOP [23].

References

1. Burke, D., Brown, K.: Efficiently handling complex local problems in distributed constraint optimisation. In: Proceedings of the European Conference on Artificial Intelligence (ECAI), pp. 701–702 (2006)
2. Dechter, R.: Bucket elimination: a unifying framework for reasoning. Artif. Intell. **113**(1–2), 41–85 (1999)
3. Fioretto, F., Le, T., Yeoh, W., Pontelli, E., Son, T.C.: Improving DPOP with branch consistency for solving distributed constraint optimization problems. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 307–323. Springer, Heidelberg (2014)
4. Fioretto, F., Le, T., Pontelli, E., Yeoh, W., Son, T.C.: Exploiting GPUs in solving (distributed) constraint optimization problems with dynamic programming. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 121–139. Springer, Heidelberg (2015)
5. Fioretto, F., Yeoh, W., Pontelli, E.: Multi-variable agent decomposition for DCOPs. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2016)
6. Geman, S., Geman, D.: Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell. **6**(6), 721–741 (1984)
7. Hamadi, Y., Bessière, C., Quinqueton, J.: Distributed intelligent backtracking. In: Proceedings of the European Conference on Artificial Intelligence (ECAI), pp. 219–223 (1998)
8. Hastings, W.K.: Monte Carlo sampling methods using Markov chains and their applications. Biometrika **57**(1), 97–109 (1970)
9. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)

10. Kumar, A., Faltings, B., Petcu, A.: Distributed constraint optimization with structured resource constraints. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 923–930 (2009)
11. Larrosa, J.: Node and arc consistency in weighted CSP. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 48–53 (2002)
12. Le, T., Fioretto, F., Yeoh, W., Son, T.C., Pontelli, E.: ER-DCOPs: a framework for distributed constraint optimization with uncertainty in constraint utilities. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2016)
13. Le, T., Son, T.C., Pontelli, E., Yeoh, W.: Solving distributed constraint optimization problems with logic programming. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2015)
14. Léauté, T., Ottens, B., Szymanek, R.: FRODO 2.0: an open-source framework for distributed constraint optimization. In: Proceedings of the Distributed Constraint Reasoning Workshop, pp. 160–164 (2009)
15. Maheswaran, R., Pearce, J., Tambe, M.: Distributed algorithms for DCOP: a graphical game-based approach. In: Proceedings of the International Conference on Parallel and Distributed Computing Systems (PDCS), pp. 432–439 (2004)
16. Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., Varakantham, P.: Taking DCOP to the real world: efficient complete solutions for distributed event scheduling. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 310–317 (2004)
17. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21**, 1087 (1953)
18. Modi, P., Shen, W.-M., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraint optimization with quality guarantees. Artif. Intell. **161**(1–2), 149–180 (2005)
19. Nguyen, D.T., Yeoh, W., Lau, H.C.: Distributed gibbs: a memory-bounded sampling-based DCOP algorithm. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 167–174 (2013)
20. Ottens, B., Dimitrakakis, C., Faltings, B.: DUCT: an upper confidence bound approach to distributed constraint optimization problems. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 528–534 (2012)
21. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 1413–1420 (2005)
22. Petcu, Adrian, Faltings, Boi V.: Approximations in distributed optimization. In: van Beek, Peter (ed.) CP 2005. LNCS, vol. 3709, pp. 802–806. Springer, Heidelberg (2005)
23. Petcu, A., Faltings, B.: MB-DPOP: a new memory-bounded algorithm for distributed optimization. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 1452–1457 (2007)
24. Ramchurn, S.D., Vytelingum, P., Rogers, A., Jennings, N.: Agent-based control for decentralised demand side management in the smart grid. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 5–12 (2011)
25. Roberts, G.O., Smith, A.F.: Simple conditions for the convergence of the Gibbs sampler and Metropolis-Hastings algorithms. Stochast. Processes Appl. **49**(2), 207–216 (1994)

26. Sanders, J., Kandrot, E.: CUDA by Example. An Introduction to General-Purpose GPU Programming. Addison Wesley, Reading (2010)
27. Shapiro, L.G., Haralick, R.M.: Structural descriptions and inexact matching. IEEE Trans. Pattern Anal. Mach. Intell. **5**, 504–519 (1981)
28. Stranders, R., Farinelli, A., Rogers, A., Jennings, N.: Decentralised coordination of mobile sensors using the max-sum algorithm. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 299–304 (2009)
29. Sultanik, E., Lass, R., Regli, W.: DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In: Proceedings of the Distributed Constraint Reasoning Workshop (2007)
30. Vinyals, M., Rodríguez-Aguilar, J., Cerquides, J.: Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. J. Auton. Agents Multi-Agent Syst. **22**(3), 439–464 (2011)
31. Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. J. Artif. Intell. Res. **38**, 85–133 (2010)
32. Yeoh, W., Yokoo, M.: Distributed problem solving. AI Mag. **33**(3), 53–65 (2012)
33. Yokoo, M. (ed.): Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems. Springer, Heidelberg (2001)
34. Zivan, R., Okamoto, S., Peled, H.: Explorative anytime local search for distributed constraint optimization. Artif. Intell. **212**, 1–26 (2014)

Morphing Between Stable Matching Problems

Ciaran McCreesh, Patrick Prosser$^{(\boxtimes)}$, and James Trimble

University of Glasgow, Glasgow, Scotland
`pat@dcs.gla.ac.uk`

Abstract. In the stable roommates (SR) problem we have n agents, where each agent ranks all other agents in strict order of preference. The problem is then to match agents into pairs such that no two agents prefer each other to their matched partners, and this is a stable matching. The stable marriage (SM) problem is a special case of SR, where we have two equal sized sets of agents, men and women, where men rank only women and women rank only men. Every instance of SM admits at least one stable matching, whereas for SR as the number of agents increases the number of instances with stable matchings decreases. So, what will happen if in SM we allow men to rank men and women to rank women, i.e. we relax gender separation? Will stability abruptly disappear? And what happens in a stable roommates scenario if agents do not rank all other agents? Again, is stability uncommon? And finally, what happens if there are an odd number of agents? We present empirical evidence to answer these questions.

1 Introduction

In the **S**table **R**oommates (SR) problem [6,7,9] we have n agents, where each agent ranks all $n-1$ other agents in strict order of preference. The problem is then to match pairs of agents in a one-one correspondence (bijection) such that the matching is *stable*. A matching is stable if there does not exist a *blocking pair* of agents ($agent_i$ and $agent_j$) in the matching such that $agent_i$ and $agent_j$ find each other *acceptable* (i.e. they rank each other) and (a) $agent_i$ is unmatched or prefers $agent_j$ to his matched partner and (b) $agent_j$ is unmatched or prefers $agent_i$ to his matched partner (see [9], 1.4.2).

The **S**table **M**arriage problem (SM) [3,4,6,9,16,17] is a specialized instance of SR where agents have gender, such that we have two sets of agents, namely men and women, both the same cardinality. A stable matching is a one-one correspondence between the men and women such that the matching admits no blocking pair (as defined above, also see [9] 1.3.4). Figure 1 shows (on the left) an instance of SR with 6 agents (sr6), and (in the middle) an instance of SM with three men and three women (sm3).

SM instance sm3 can be represented as an instance of the Stable Roommates Problem with Incomplete lists (SRI), i.e. we have incomplete lists where some

agents find each other unacceptable (see [9], 1.4.2). This is shown on the right of Fig. 1 (sri6), where agents 1 to 3 represent the men in sm3, and agents 4 to 6 represent the women in sm3. Therefore the matching $\{(1,1),(2,2),(3,3)\}$ in sm3 corresponds to the matching $\{(1,4),(2,5),(3,6)\}$ in sri6.

```
1: 6 3 5 2 4
2: 3 5 1 6 4                                      1: 4 5 6
3: 2 6 1 5 4        1: 1 2 3    1: 1 2 3          2: 5 6 4
4: 5 1 2 3 6        2: 2 3 1    2: 1 3 2          3: 4 6 5
5: 6 1 2 3 4        3: 1 3 2    3: 3 1 2          4: 1 2 3
6: 4 2 5 1 3                                      5: 1 3 2
                                                  6: 3 1 2
```

Fig. 1. A Stable Roommates (SR) instance on the left sr6, with 6 agents. In the middle an instance of Stable Marriage (SM) with three men and three women, sm3. On the right, sm3 is represented as an instance of Stable Roommates with Incomplete lists (SRI), instance sri6. Instance sr6 has two stable matchings, namely $\{(1,5),(2,3),(4,6)\}$ and $\{(1,4),(2,3),(5,6)\}$. Instance sm3 has a single stable matching $\{(1,1),(2,2),(3,3)\}$. Instance sri6 has only one stable matching $\{(1,4),(2,5),(3,6)\}$ and this corresponds to the stable matching for sm3.

The underlying structure of SR is a simple graph [10], where an edge corresponds to a pair of agents who find each other acceptable. The SR graph is therefore a clique K_n and preference lists of agents are permutation of their adjacency lists. The SM graph is a complete biclique, with two sets of vertices (men and women), each of size $n/2$ with $n^2/4$ edges. In SRI the underling graph is again a simple graph, but not complete. Every instance of SM admits at least one stable matching [3], but this is not the case for SR [7], where some instances admit no stable matching (the same is true of SRI). As the number of agents increase the proportion of SR instances with stable matchings falls [7,11,12,14,15].

Therefore, we have a spectrum of stable matching problems. At one extreme we have SM, highly structured (a biclique), always with stable matchings and at the other extreme we have SR, unstructured, with a falling number of stable matchings as the number of agents increases [3,6,7,9]. Therefore, what happens as we replace some of the structure in SM with randomness from SR, i.e. what happens when we blend SM and SR? Will there be an abrupt change in behaviour, where the average number of instances with stable matchings falls, or will it be smooth and gradual with stability declining slowly?

There is also a spectrum of stable roommates problems. At one extreme we have SRI instances with empty preference lists, where every agent finds every other agent unacceptable, and corresponds to the edgeless graph. This instance has one stable matching, where every agent is happy to be unmatched[1]. At the other extreme we have SR, where every agent finds every other agent acceptable,

---

[1] We assume that every agent ranks himself in last position and can potentially be self-matched.

not always admitting a stable matching, and this corresponds to the complete graph. Therefore we can move across this spectrum, gradually increasing the number of acceptable pairs (edges in the graph). As we do so, will the proportion of instances with stable matchings fall gradually, or will it fall abruptly?

And finally, it is tacitly assumed that the number of agents is even. Why should that be so? Imagine we had a conference where delegates share rooms in the students' halls of residence, two to a room. Would it be possible to limit attendance to only an even number of delegates? And if delegates rank one another in order of preference, is it more likely to be an unstable scenario when the number of delegates is odd? We investigate this also.

In the following section we describe how we can mix SM and SRI in a controlled manner (problem generation). We then present empirical results for mixing SM and SRI followed by experiments on gradually moving from SRI with no acceptable pairs to SR, where the number of agents is even and when the number of agents is odd.

2 Problem Generation

Given a graph $G = (V, E)$, where V is the set of vertices and E the set of edges, we can create a stable matching problem from G as follows. The set of vertices correspond to the set of agents and an edge (i, j) is in E if and only if $agent_i$ and $agent_j$ are an acceptable pair (i.e. they rank each other). Assume vertex neighbourhood is represented as an adjacency list. Given an edge $(u, v) \in E$, add u to the list $adjacent[v]$ and add v to the list $adjacent[u]$. Once this has been done for all edges, perform a Knuth shuffle[2] [2] on each adjacency list and treat these as preference lists. This is essentially the technique used in Sect. 2 of [10].

We now describe two techniques for creating a blended graph, with m edges, from two input graphs. Consider simple graphs G_1 and G_2, both of the same order n (number of vertices), where G_1 is the complete bipartite graph $K_{n/2,n/2}$ and G_2 is the clique K_n. Assume we have a mixing proportion p, where $0 \leq p \leq 1$. We can mix these two graphs to produce a new graph G_3 with $(1 - p)m$ edges taken from G_1 and $p.m$ edges taken from G_2. This can be done in two ways. The first is similar to the rewiring technique of Watts and Strogatz [8]. This is presented in Algorithm 1 and will be refered to as *model A*. The algorithm returns a set of edges E, i.e. acceptable pairs, where that set is of size $m = n^2/4$, the same size as the biclique.

Algorithm 2 corresponds to the type-B morph described in [5], where again $m = n^2/4$ edges are to be produced (line 3). The set E_1 contains m randomly selected edges from K_n, the set E_2 contains all edges in $K_{n/2,n/2}$ and set E is the intersection of these two sets (lines 4, 5 and 6). The remaining number of edges to be added to E is $\delta = m - |E|$, where $p.\delta$ edges are randomly selected

---

[2] To permute an array of n elements, vary i from n down to 2, randomly select j in the range i to 1 inclusive, then swap the ith and j^{th} array elements.

Algorithm 1. modelA: select $m.(1-p)$ from biclique and $m.p$ from clique

---
1 $\mathbf{Set}\langle\mathbf{Edge}\rangle$ modelA(**int** n, **real** p)
2 **begin**
3 $\quad$ **int** $m \leftarrow n^2/4$
4 $\quad$ $E_1 \leftarrow \{(i,j) \mid 1 \le i < j \le n\}$
5 $\quad$ $E_2 \leftarrow \{(i,j) \mid 1 \le i \le n/2 \ , \ n/2 < j \le n\}$
6 $\quad$ $E \leftarrow select((1-p).m, E_2)$
7 $\quad$ $E \leftarrow E \cup select(p.m, E_1 \setminus E)$
8 $\quad$ **return** E

Algorithm 2. modelB: a type B morph

---
1 $\mathbf{Set}\langle\mathbf{Edge}\rangle$ modelB(**int** n, **real** p)
2 **begin**
3 $\quad$ **int** $m \leftarrow n^2/4$
4 $\quad$ $E_1 \leftarrow select(m, \{(i,j) \mid 1 \le i < j \le n\})$
5 $\quad$ $E_2 \leftarrow \{(i,j) \mid 1 \le i \le n/2 \ , \ n/2 < j \le n\}$
6 $\quad$ $E \leftarrow E_1 \cap E_2$
7 $\quad$ $\delta \leftarrow m - |E|$
8 $\quad$ $E \leftarrow E \cup select(p.\delta, E_1 \setminus E) \cup select((1-p).\delta, E_2 \setminus E)$
9 $\quad$ **return** E

---

from $E_1 \setminus E$ and $(1-p).\delta$ edges are randomly selected from $E_2 \setminus E$. Therefore when $p = 0$ both models deliver an instance of SM, and when $p = 1$ both deliver an instance of SRI with $p.n^2/4$ edges drawn at random from K_n.

3 The Empirical Study

The majority of the study used a constraint programming formulation of the stability constraint proposed in [15]. In all our models each agent ranks himself in last position. Consequently an agent can self-match, if and only if this results in stability. The exp eriments were run on an Intel Xeon CPU E5-2660 processor at $2.2\,\mathrm{GHz}$ with $20\,\mathrm{Mb}$ of cache and $128\,\mathrm{Gb}$ of RAM. In many of the studies the control parameter is p (as an edge probability or mixing proportion), and is varied in steps of 0.001 with a sample size of 1,000.

3.1 Morphing from SM to SRI

The first experiment investigates what happens as we morph from SM to SRI using models A and B, and what happens as we increase the number of agents. This is shown in Fig. 2. On the x axis we have p, the mixing parameter, and when $p = 0$ all instances are bipartite and when $p > 0$ instances are non-bipartite. On the y axis we have the average percentage of instances that admit a stable matching, i.e. the percentage that were satisfiable. On the left we have two contours, both for

$n = 100$, one for model A the other for model B. This shows that there is only a small difference in the behaviours of the two models, i.e. they produce similar behaviour, with model B preserving the SM properties (bipartite) slightly longer than model A.

On the right are contours for n varying from 50 to 400, using only model A. At $p = 0$ all instances are satisfiable, as expected, but what is surprising is how rapidly behaviour changes with a small degree of mixing and that this becomes more abrupt as the number of agents increases. Although not shown, as p increases, the number of stable matchings per instance falls rapidly. In summary, a small degree of within-gender acceptability results in a rapid loss of stability.[3]
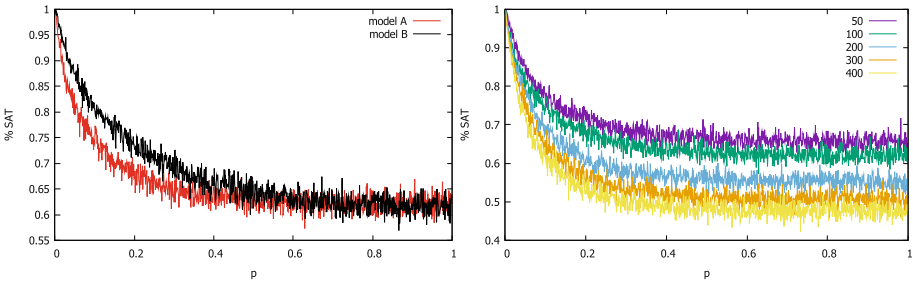
Fig. 2. Morphing from SM to SRI. The x-axis is p, the mixing proportion, and the y-axis is the percentage of instances admitting a stable matching. On the left, model A and model B with $n = 100$. On the right, contours for a variety of n, 50 to 400.

3.2 Morphing Between SR and SRI

In [10] Mertens empirically investigated SR (measuring P_n, the probability of a stable matching existing in K_n), SRI where agents exist on a grid (and rank only their Moore neighbourhood) and SRI on graphs with a given average degree (where random graphs were generated with increasing order but with an average degree of only 35, 45 or 60). We now investigate what happens in the stable roommates problem as we vary the amount of acceptability between agents, i.e. we vary average degree. Again, viewing the problem as a simple graph, we vary the edge probability p and with that the degree of the graph, and this corresponds to the average length of preference lists. When edge probability is one every agent finds every other agent acceptable and problem instances are SR, when p is zero every agent finds every other agent unacceptable and agents are happy to be alone (i.e. self-matched), and when $0 < p < 1$ preference lists are incomplete and we have SRI instances. Figure 3 shows, on the left, contours for $n = 100$ and $n = 101$, with probability of acceptability on the x axis and

---

[3] We leave any social interpretation of these observations to others.

percentage of stable instances on the y axis. On the right is the average size of a stable matching when one exists[4], and it should be noted that for a given instance all its matchings are the same size [16].
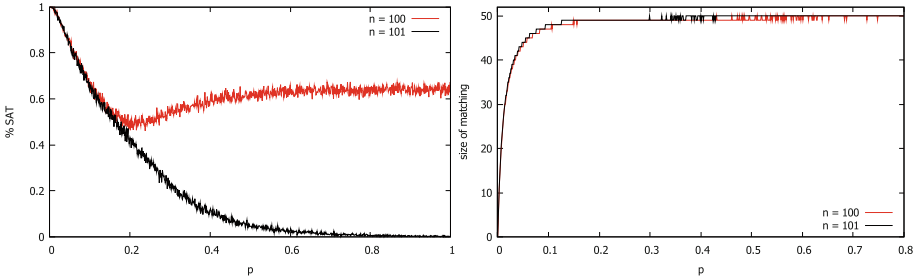
Fig. 3. Increasing the proportion p of agents found to be acceptable in stable room-mates. On the x axis p, probability that two agents find each other acceptable. On the left, the y axis is percentage of instances that admit a stable matching. On the right, y axis is average size of stable matching when one exists. Note that when an instance admits a stable matching, all its stable matchings are the same size [6]. Contours are shown for even ($n = 100$) and odd ($n = 101$) number of agents.

The shape of these curves are surprising. All instances are stable when $p = 0$ (i.e. all agents are happy to be alone) and this then falls away. Tabulated results for SRI predict that this will happen [7,11–15]. But what was not predictable was the shape of the $n = 100$ contour: falling sharply, climbing abruptly and then tapering off to its final value when SRI becomes SR. Note that the "knee" in the contour on the left of Fig. 3 comes some time after the point where random graphs $G(n, p)$ become a single component. Therefore the "knee" is not due to the emergence of a giant component.

When n is odd there are indeed stable matchings in our model (an odd number of agents must self-match), but these stable matchings are typically either small or rare. However, the shape of the odd contour on the left is perhaps not so surprising. For $n = 101$, suppose all preference lists are complete, and without loss of generality suppose that agent 101 is unmatched. Then for agent 101 not to block the matching, every other agent must have a better partner than him, which is unlikely to be satisfied in general. Obviously the longer the preference lists in general, the more chances agent 101 has to block the matching, hence the shape of the $n = 101$ contour (on the left)[5].

Figure 4 separates even (contours on the left) from odd (contours on the right) for various number of agents. This time the x axis is $n.p$ and that is the average degree of the underlying graph, and is then the average size of preference lists. For compactness, the x-axis is shown on a log scale.

---

[4] The x axis is cut short due to small sample size for odd n and large p.
[5] We thank David Manlove for this explanation.
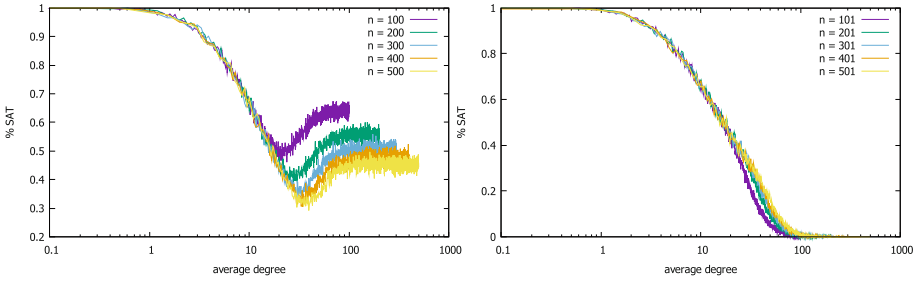
Fig. 4. Percentage of SRI instances with stable matchings as we vary probability that agents find each other acceptable. On the left n is even and on the right n is odd. The x-axis is a logscale of $n.p$, i.e. average degree.
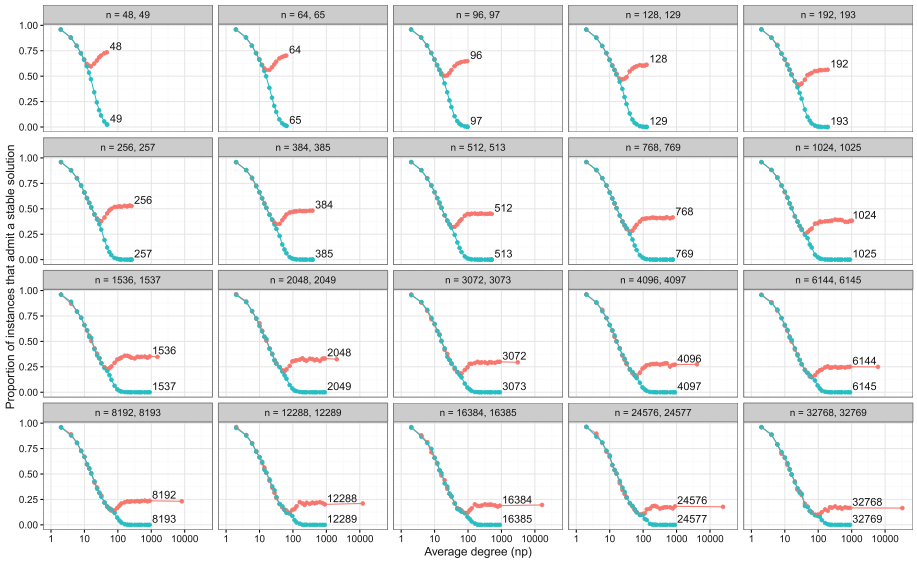
Fig. 5. Increasing the proportion p of agents found to be acceptable. On the x axis the log of average degree $(p.n)$, the y axis is percentage of instances that admit a stable matching. The red contour is for an even number of agents n and the blue contour for odd number of agents $n + 1$.

The experiments were then repeated for larger values of even n and odd $n+1$. Graphs were generated using the algorithm of Batagelj and Brandes [1], which allows much faster generation of large, sparse graphs than simple quadratic-time methods. Each instance was solved using both Irving's algorithm [7] and a simple SAT encoding using MiniSat, to verify correctness of our implementation. Sample size varied from 2,000 to 1,000,000. We ran our program for values of np up to 896; these results are shown in Fig. 5. In addition, for large even values of n the figure shows results for $p = 1$ from [12]; these appear as the rightmost

points. It appears that when n is even and n increases, its odd partner *hugs* onto it for longer and longer, leading us to expect that in the limit both curves will be indistinguishable. That is, when there are a large number of agents it will not matter if that number is odd or even.

4   Conclusion

Stable marriage problems always admit a stable matching but this is not true for stable roommates. In our experiments, a certain amount of disorder was added to SM such that we permit within-gender matching. This brought about an abrupt change in the behaviour of the problem, with an abrupt fall in the proportion of instances with stable matchings.

   In the roommates problem, with a small number of agents, it appears to matter if the number of agents is odd or even. If the number of agents is odd, stable matchings tend to be small or scarce. When the number of agents is even, the proportion of instances with stable matchings falls as acceptability increases then abruptly climbs, and this climb then gradually tapers off as the instances tend to SR with complete preference lists. However, when n is large, we conjecture that behaviour will tend to be that observed when n is odd.

References

1. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Phys. Rev. E **71**, 036113 (2005)
2. Durstenfeld, R.: Algorithm 235: random permutation. CACM **7**, 420 (1964)
3. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)
4. Gale, D., Sotomayor, M.: Some remarks on the stable matching problem. Discrete Appl. Math. **11**, 223–232 (1985)
5. Gent, I.P., Hoos, H.H., Prosser, P., Walsh, T.: Morphing: combining structure and randomness. In: Proceedings of the Sixteenth National Conference on Artificial Intelligence, pp. 654–660 (1999)
6. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. The MIT Press, Cambridge (1989)
7. Irving, W.R.: An efficient algorithm for the stable roommates problem. J. Algorithms **6**(4), 577–595 (1985)
8. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. Nature **393**, 440–442 (1998)
9. Manlove, D.: Algorithmics of Matching under Preferences. Theoretical Computer Science, vol. 2. World Scientific, Singapore (2013)
10. Mertens, S.: Random stable matchings. J. Stat. Mech. Theory Exp. (10), P10008 (2005)
11. Mertens, S.: Small random instances of the stable roommates problem. J. Stat. Mech. Theory Exp. **2015**(6), P06034 (2015)

12. Mertens, S.: Stable roommates problem with random preferences. J. Stat. Mech. Theory Exp. **2015**(1), P01020 (2015)
13. Ng, C., Hirschberg, D.S.: Lower bounds for the stable marriage problem and its variants. SIAM J. Comput. **19**, 71–77 (1990)
14. Pittel, B., Irving, R.W.: An upper bound for the solvability of a random stable roommates instance. Random Struct. Algorithms **5**(3), 465–487 (1994)
15. Prosser, P.: Stable roommates and constraint programming. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 15–28. Springer, Heidelberg (2014)
16. Roth, A.E.: The evolution of the labor market for medical interns and residents: a case study in game theory. J. Polit. Econ. **92**(6), 991–1016 (1984)
17. Roth, A.E., Sotomayor, M.A.O.: Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis. Econometric Society Monographs, vol. 18. Cambridge University Press, Cambridge (1990)

Testing and Verification Track

Using Graph-Based CSP to Solve the Address Translation Problem

Merav Aharoni[1], Yael Ben-Haim[1(✉)], Shai Doron[1], Anatoly Koyfman[1],
Elena Tsanko[2], and Michael Veksler[3]

[1] IBM Research, Haifa, Israel
{merav,yaelbh,dshai,anatoly}@il.ibm.com
[2] IBM Systems, Austin, TX, USA
etsanko@us.ibm.com
[3] Haifa, Israel
mickey.veksler@gmail.com

Abstract. The hardware address translation mechanism is an essential part of modern microprocessor memory management. The ever-growing demand for performance and low power of integrated circuits makes this mechanism exceptionally complex, and its verification requires sophisticated test generation tools. This paper presents a solution, based on constraint satisfaction, to generate stimuli for testing address translation.

The address translation process passes through a sequence of steps and can therefore be naturally described as a directed acyclic graph. We developed a framework that we call *graph-based constraint satisfaction problems (GCSP)*. These problems consist of a directed graph, combined with a CSP, where each variable and constraint of the CSP is linked to a particular node or edge of the graph. A solution to the problem is a path in the graph, such that all constraints defined along this path must be satisfied. We base our algorithm for solving GCSPs on conditional CSP. We successfully used this technology to verify the memory management units of several industrial microprocessors.

Keywords: Constraint programming · Graph · Address translation · Memory management unit · Processor verification · Conditional CSP · Network · Path · Dynamic CSP · Graphplan · Planning

1 Introduction

Hardware designs are most commonly verified by generating stimuli using dedicated test generators. Many of these test generators use constraint solvers to solve complex constraint satisfaction problems (CSPs) [18]. One of the most

complex mechanisms in modern designs is the address translation mechanism [5,20], which is part of the memory management unit (MMU). The mapping of addresses from virtual memory to physical memory passes through several stages and access tables. There are usually several alternative mappings depending on the application, state of the machine, page size, memory region and more. Thorough verification of this mechanism requires sophisticated test-generation methods [15].

To solve the address translation test-generation problem, we modeled it as a *graph-based constraint satisfaction problem* or GCSP. A GCSP consists of a graph combined with a constraint satisfaction problem (CSP), such that each of the variables and constraints of the CSP are linked to particular nodes and edges of the graph. A solution to a GCSP is a subgraph, along with a solution to a sub-CSP consisting of those variables and constraints that are linked to edges and nodes that belong to the subgraph.

In this paper, we discuss directed graphs with a single source and target, where the solution is a path from source to target. We show how this concept is used in IBM to generate tests for the verification of address translation units in several designs, for a variety of architectures, including IBM® Power Systems^{TM}[1], IBM® z Systems^{TM}, and ARM. The concept is generic and elementary and can be applied to other problems that can be modeled as a selection of a random path in a graph, where every path is subject to given constraints.

The novel contribution of this paper is:

1. Modeling the address translation test-generation problem as a graph-based CSP.
2. GCSP is converted internally to activity CSP [13], a variant of conditional CSP, by translating graph constraints to activity variables and constraints. This enables efficient solving of the GCSP.
3. Solving the challenging problem of generating constrained-random test-cases for verifying the address translation mechanism.

The remainder of this paper is organized as follows: in the next section, we provide a formal definition of graph-based CSP along with an example. Section 3 presents related work from the domain of constraint satisfaction. Sections 4 and 5 describe the address translation test-generation problem and its modeling as a GCSP respectively. In Sect. 6, we offer a brief background on activity CSP, a variant of conditional CSP, based on [13]. In Sect. 7, we explain how we translate a GCSP to an activity CSP. Section 8 presents our results from industrial applications. We conclude in Sect. 9.

---

[1] IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

2 Graph-Based Constraint Satisfaction Problems

In the definition that follows, we use the expression *graph entity* to refer to a node or an edge, since we will often refer to both entities in the same context.

We define a graph-based constraint satisfaction problem (GCSP) as follows:

Definition 1. *$GCSP = \langle V, D, C, G \rangle$ is a constraint satisfaction problem $CSP = \langle V, D, C \rangle$, combined with a directed graph $G = \langle N, E \rangle$. Every variable $v \in V$ and every constraint $c \in C$ is linked to a particular node $n \in N$ or to an edge $e \in E$ in the graph. Thus CSP variables and constraints are linked to graph entities. D denotes the variables' initial domains.*

A solution to a GCSP is a subgraph $G' = \langle N', E' \rangle \subseteq G$, where $N' \subseteq N$ and $E' \subseteq E$, and an assignment to the variables linked to graph entities in G', such that all constraints linked to graph entities in G' are satisfied.

Note the following:

1. Variables and constraints that are not linked to graph entities in G' are ignored.
2. Although in the above definition each variable and constraint is linked to a graph entity, in practice we actually allow some of the variables and constraints to be unlinked to graph entities. These variables and constraints must be satisfied in any solution. To simplify the discussion, we will assume no such variables and constraints. There is no loss of generality, because we can link these to a node that must be part of any solution.
3. If a constraint has one or more parameters that are linked to a graph entity that is not part of the solution, the constraint need not be satisfied. Specifically, a constraint must be satisfied if its parameters and the constraint itself are all linked to graph entities that are part of the solution.

For the purposes of this paper, we focus on a particular type of graphs and solutions: we are interested in directed graphs with a single source and target. We are looking for solutions comprising a single path from source to target in this graph. We note that the proposed solution is applicable to any directed graph, although in the application of address translation, the graph is always acyclic.

We demonstrate the concept of a GCSP with the following example.

Example 1. The graph in Fig. 1 consists of four nodes, n_s, n_1, n_2, n_t. The goal is to select a path in the graph from n_s, the source node, to n_t, the target node. Thus n_s and n_t are part of any solution. There are two possible paths in the graph: $p_1 = (n_s, n_1, n_t)$ and $p_2 = (n_s, n_2, n_t)$. We define four CSP variables in this GCSP: x, y, z, w, each with initial domain $\{1, 2, 3, 4\}$. These variables are linked to respective nodes n_s, n_1, n_t, n_2, as portrayed in Fig. 1. We define four constraints in this problem, each linked to an edge of the graph. If p_1 is selected, then variable y must be in the solution, and constraints $y = x + 1$ and $z = y + 2$ must be fulfilled. Thus a possible solution is $x = 1, y = 2, z = 4$. Note that w and

its related constraints are ignored. If p_2 is selected, variable w must be part of the solution, and constraints $w = x - 2$ and $z = w$ must hold. Possible solutions are $x = 4, w = 2, z = 2$ and $x = 3, w = 1, z = 1$.

We now assume that the initial domains of all variables are changed to $\{1, 2, 3\}$. In this case, p_1 is not possible, and the only solution is p_2, with $x = 3, w = 1$, and $z = 1$. This example demonstrates that propagation can occur from the CSP to the graph entities. If a domain becomes empty during propagation, it may force the exclusion of a particular edge or node from the solution path.
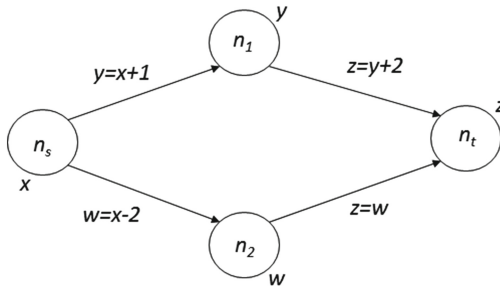
Fig. 1. Example of a simple GCSP

3   Related Work

We cite some of the most relevant research on combining graphs into CSPs. Fages, in his Ph.D. dissertation [12], provides an excellent survey of the existing work. Le Pape et al. [19] define a graph extension to constraint programming, which introduces variables that represent a path in a graph. The graph here is weighted and the focus is on routing in networks with optimization, where the goal is to find the shortest path. Dooms et al. [11] describe several propagators on graph variables, in particular a path propagator. Cambazard and Bourreau [9] also discuss a path propagator and model the graph with a Boolean variable for every edge.

In our model, we do not explicitly define a graph variable, but rather a Boolean variable for each graph entity. Our solution approach is based on activity CSP, and maps graph entities to activity variables, as shown in Sect. 7.

Many of the problems in the literature can be described as seeking a path in a graph. Problems of this type include configuration and planning problems [21]. The notion of *planning graphs*, often referred to as *Graphplan*, was first introduced by Blum and Furst [8] and is often used to describe problems of this category.

A planning graph is a method to represent an action plan. It can be described intuitively in the form of a level graph, where the nodes (proposition lists) represent states of a system, and the edges (action lists) represent transitions between

states. An initial level represents an initial state and a final level represents the final state(s). Each state has pre-conditions and actions have post-conditions. A path from the initial state to the final state represents a particular choice of plan, such that all actions along this path are taken, and all conditions along this path hold. Planning graphs are often solved using dynamic CSP [17], as shown in [10].

A graph-based approach to create stimuli for verification is also used by Hamid [14]. This work defines target nodes that correspond to verification goals. Paths from a single source to the target nodes define scenarios to achieve the verification goals. The emphasis in Hamid's work is on selecting the correct path toward hitting a particular test case by inspecting the graph. It does not mention allowing constraints on the various paths in the graph.

Our solution approach differs from dynamic CSP because we represent the entire problem graph from the start, and do not build the solution gradually. There are two advantages to our approach. The first is the ability to reach a more uniform distribution of the solution space. The second is that constraints and conflicts appearing "late" along a solution path, can in our approach, be propagated early, to affect the possible paths chosen, thus avoiding backtracks. The drawback of our approach is that the entire problem must be represented at once. For huge problems, this may be impossible.

4   Background on Address Translation

The memory management units (MMU) of modern hardware architectures provide a virtual address space for their applications. The operating system uses address translation mechanisms to map the virtual addresses used by a program to actual hardware resources. The operating system maintains these resources through software, and the hardware carries out the translation itself. Address translation provides several benefits, such as allowing the applications to be oblivious of the actual memory accesses, higher security by isolating applications that use shared memory, and conceptually increasing the memory size by paging. For more information about memory management and address translation specifics see [7].

The ever-growing demand for performance causes these translation mechanisms to become more complex, thereby increasing the risk for manufacturing defects. Traditionally, simulation-based techniques have played a major role in functional verification. Current practices foster the use of automatic constrained-random stimuli generators to provide high quality tests. The verification engineer has a full range of control over the generated test, from fully constrained to fully random [18]. These generators typically consist of a model for the architecture, a general purpose test generator, and an internal constraint solver. The test generator receives as input test templates that describe the particular scenario for which the verification engineer wishes to create a test instance. The constraints stemming from the architecture model along with the test template are combined to create a constraint satisfaction problem. The constraint solver creates a random instance, which is the desired test.

Several special features are required from a constraint solver used for hardware verification. The first is generation of uniformly distributed random solutions. Although it is sufficient to generate a single solution in many types of CSPs, in the domain of verification it is essential to generate many random solutions for any problem instance. This is because no one can predict where the bugs may reside. The second feature is the support for CSP variables of type bit vectors. These are used to represent registers and memory locations and may be 64 or 128 bits wide. Support for this data type includes special propagators such as addition, concatenation, and bit operations between bit vectors. A third important feature is support for soft constraints. Soft constraints add higher probability of hitting interesting corner cases that were not explicitly defined by the user. The solver will try to add these cases when possible, but will not fail if these cannot be generated.

Address translation is one of the most complex mechanisms in processor design. For verification purposes, we are interested in scenarios such as targeting a certain physical address, covering all types of walks among the translation tables, or targeting corner cases. This led to the development of specialized address translation generation schemes as part of the general purpose test generators. Kornykhin [16] describes an algorithm to generate interesting tests for cache events and address translation. His approach is to define the interesting cache events, such as miss and hit, and to generate tests to create the required events. To our understanding, this approach is more limited in scope than ours. For example, one cannot define a constraint on the physical address. The work described in our paper is based on an earlier version, described by Adir et al. [4,5]. They too use conditional CSP to solve the problem. However, they did not explicitly define the graph as a separate entity of the CSP. Separating the graph entities from the rest of the CSP allowed us to create a much simpler, more general and maintainable model.

The address translation scheme can be described as a directed acyclic graph (DAG). Each node in the DAG represents a step in the translation. The edges represent transformations that occur between steps, and are determined by the processor state. A translation instance will be a path along this DAG, such that all constraints introduced by the overall processor state are satisfied. This problem introduces a special challenge to a constraint solver; constraints along a given path have to be satisfied if and only if this path is indeed selected for the solution instance.

5   Modeling Address Translation as Graph-Based CSP

In this section, we give a basic description of the translation process in ARM architecture. The translation process is similar in other architectures, and we refer the user to [7] for an in-depth explanation of this process. To verify the translation mechanism using test generation, we model the translation as a graph-based CSP. To generate interesting stimuli, we constrain various elements in this model to generate relevant tests cases. In the scope of this paper, we present the idea of how this graph-based CSP is constructed. For a more detailed description of the model, please refer to this paper's supplementary material.

The ARM virtual memory system architecture (AArch64 VMSAv8-64) defines a table lookup scheme for the translation of virtual address (VA) to physical address (PA) [2]. Addresses are translated in chunks of pages, each one a consecutive address space of a fixed size. The smallest resolution for addresses is a single byte. The physical address marks the beginning of the page. The scheme is divided into two translation stages known as stage-1 and stage-2. These two stages can be nested, so that stage-1 translation invokes stage-2 translation. The architecture allows a variety of VA and PA sizes, and a lookup scheme with tables of different sizes: 4 KBytes, 16 KBytes or 64 KBytes (a.k.a. Granule size). In addition, there are several different options of page sizes. The following example shows a single stage translation from a 48 bit VA to a 48 bit PA where the granule size and page size are both 4 KB. Although in the actual solution, we handle all the specified combinations, for the sake of simplicity we limit ourselves in this paper only to a single stage of 4 KB granule. Figure 2 outlines the translation sequence from a virtual address to a physical address.
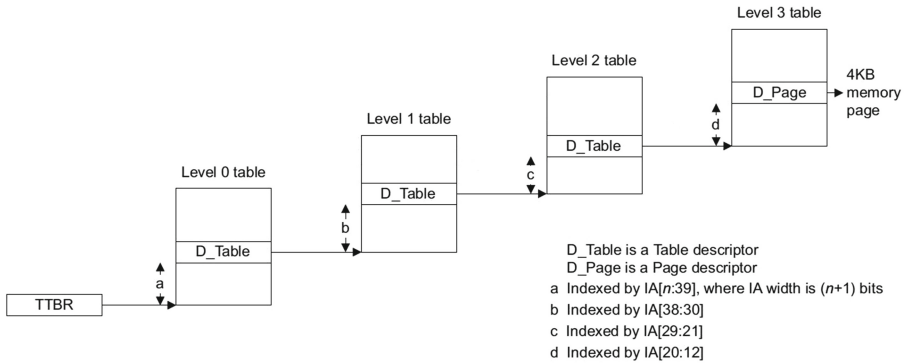
Fig. 2. Example for ARM translation scheme

For our discussion, the MMU consists of:

1. 4 *translation tables*, level 0 to level 3, each of size 4KB. These tables are divided into 512 entries of size 8 bytes each.
2. A 64 bit wide *translation table base register* (TTBR), which contains the location of the level 0 table in memory. The TTBR is aligned to 4KB; in other words, its lower 12 bits are all 0.
3. Various registers that affect the *context* of the translation, such as what region of physical memory can be accessed (e.g., secure or non-secure), what page size to use, and more.

Given a 48 bit VA, the MMU returns a PA as follows: The VA is divided into 5 fields, of width 9, 9, 9, 9, and 12 bits, which we refer to as VA[47..39], VA[38..30], VA[29..21], VA[20..12] and VA[11..0], respectively. Bits VA[47..39] indicate the table entry to be selected in the level 0 table. Note that 9 bits encode 512 entries.

Since these entries are 8 bytes wide, they are encoded by 12 bits, the top 9 bits coming from VA[47..39] and the lower 3 being all 0. Consequently, the address of this entry is computed by a bit-or operation between the TTBR and these 12 bits.

The 64 bit entry selected from the level 0 table serves as the base address for the level 1 table. An entry in the level 1 table is selected in a similar manner using VA[38..30]. The process is repeated for levels 2 and 3. Finally, the entry selected from the level 3 table serves as the base address for the designated 4KB page in the physical memory space. The actual address in memory is accessed by VA[11..0]. The address is computed by bit-or between the 64 bit entry from the level 3 table with VA[11..0]. Note that although all entries are 64 bits wide, physical addresses are 48 bits wide and the base addresses must be aligned on the 12 lower bits for 4 KB tables (or pages). Therefore, only 36 bits in the entry are needed to indicate a base address, and the remaining 28 bits in the entry are used for other means and ignored for the address computation.

The above describes the basic lookup sequence for traversing from level 0 to level 3. The general scheme also allows starting directly from level 1 or from level 2 (Table Concatenation) and ending at level 1 or at level 2 (Block Translation). These translation paths can support VA sizes smaller than 48 bit and produce translations of different page sizes.
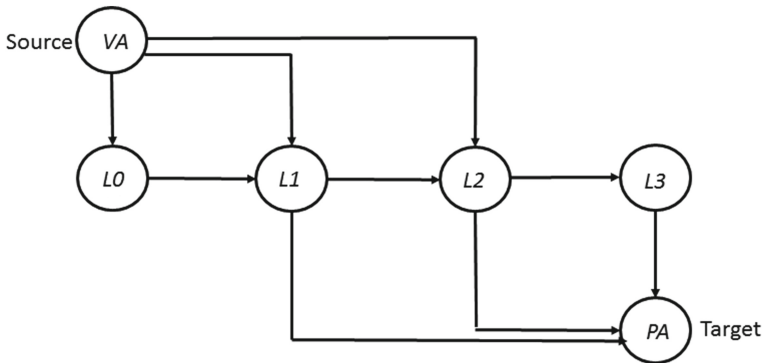
Fig. 3. Example for graph-based CSP model for ARM translation scheme

Figure 3 shows the graph constructed to model the translation. Our graph-based model for this problem consists of:

1. A node for the VA (source)
2. A node for the PA (target)
3. A node for each table at levels $x, x = 0, 1, 2, 3$
4. An edge from VA to table at levels $x, x = 0, 1, 2$
5. An edge from level x table to PA, $x = 1, 2, 3$
6. An edge from level x to level $x + 1, x = 0, 1, 2$

The CSP consists of the following:

1. For each level x node, $x = 0, 1, 2, 3$, we have the following variables:
   (a) Base address of table
   (b) Address of entry in table
   (c) Data of entry in table
   Each of the above variables is of type 64-bit wide bit-vector. The initial domain may be universal, or some subset, depending on user inputs and previous translations. For example, physical addresses are 48 bits wide and entry addresses are aligned to 8-bytes (lower 3-bits are 0). Therefore, the initial domain of entry addresses is `0x0000_XXXX_XXXX_XXX8` in hexadecimal notation, where `X` signifies a *don't care*.
2. Additional variables that represent the state of the processor (set of registers) and a snapshot of physical memory where the translation tables are allocated. These variables are not linked to any specific graph entities. Because the full description of the translation tables is huge, we only need to book-keep entries that have already been initialized in previous translations, and entries that are affected by the current translation. Note that during test generation, the program may generate hundreds or thousands of translations. Every subsequent translation must be consistent with previous translations.
3. For every $x = 0, 1, 2, 3$, we have the following constraints linked to each edge from level x to level $x + 1$:
   (a) A constraint to calculate address of entry in the table at each level. For example, the constraint to calculate the entry address in the table at level 1 is:
   ```
   FirstLevelEntryAddress = (FirstLevelBaseAddress bit_or
   shiftl(extend(sub_field(VirtualAddress,38,30),ZERO),3))
   ```
   where we use the following propagators, operating on bit vectors: $\text{sub\_field}(A, i, j)$ refers to bits i to j of variable A; $\text{extend}(A, ZERO)$ means extend A with zeros to the left; $\text{shiftl}(A, k)$ takes the variable A shifted left by k bits, filling in 0s on the right; $A = B$ bit_or$C$ means A is the bitwise-or between B and C. Note that all of the above are CSP propagators, affecting the domains of CSP variables A, B, and C. In this context, i, j and k are indices, not CSP variables.
   (b) A constraint to synchronize the data in entries of table at level x with physical memory.
   (c) Similar constraints as above for the edges VA to level x for $x = 0, 1, 2$, and for the edges level x to PA for $x = 1, 2, 3$.

In addition to the basic model, the user may add constraints to specify particular attributes of the desired solution. These can be hard or soft constraints. For example, the user can specify the desired page size. The user can also request access to a certain area of the memory by defining part of the bits in VA or PA, or request access to a certain entry in one of the tables. In practice, users create tests of thousands of assembly instructions, with many translations. Although

each translation is solved in a separate CSP, existing translations add complexity to the available memory space and to the available entries because each translation must be consistent with previous translations.

Soft constraints are generally used by the user to randomly insert more direction toward particular corner cases. One such example is to request some portion of the translations to share part of the translation tables with previous translations, but not the entire path. Another example is to invoke particular exceptions (interrupts) at a high rate.

6 Background on Activity CSP

In Sect. 7, we show how to translate GCSP to activity CSP (ACSP). We give an introduction to ACSP in this section. See [13] for more details[2].

ACSP is a variant of conditional CSP [21]. Informally, ACSP is a CSP where some Boolean variables, called *activity variables*, are special; the existence of parts of the CSP depends on the value assigned to the activity variables. We illustrate this concept using the following example:

Example 2. An ACSP consisting of:

- Six variables a, a_1, a_2, v, v_1, v_2. The initial domain of a is $\{True\}$, of a_1 and a_2 is $\{True, False\}$, and of v, v_1, v_2 is $\{1, 2, 3, 4\}$.
- a_1 and a_2 are activity variables. As for the rest of the variables, v is linked to a, v_1 is linked to a_1, and v_2 is linked to a_2.
- Two constraints c_1 and c_2. Constraint c_1 is: $v > v_1$, constraint c_2 is: $v < v_2$. c_1 is linked to a_1 and c_2 is linked to a_2.

Examples of possible solutions:

- $a = True$, $a_1 = True$, $a_2 = True$, $v = 2$, $v_1 = 1$, $v_2 = 3$ ($a_1 = True$, therefore v_1 participates in the solution, and c_1 must be satisfied. Similar for a_2).
- $a = True$, $a_1 = True$, $a_2 = False$, $v = 2$, $v_1 = 1$ ($a_2 = False$, therefore v_2 does not participate in the solution, and c_2 does not have to be satisfied).
- $a = True$, $a_1 = False$, $a_2 = False$, $v = 3$.

To simplify, we assume that every variable and constraint is linked to an activity variable. If we want a variable to be independent, and unlinked to any graph entity, we can create a dummy activity variable whose initial domain is $\{True\}$, and link the variable to it. In example 2, v is such a variable and it is linked to activity variable a.

---

[2] The definition for ACSP that we provide here is slightly different, yet equivalent to the definition in [13].

6.1 Formal Definition of ACSP

An ACSP is defined as a tuple $\langle V, V_A, D, C, M \rangle$, where:

- V is the set of variables.
- $V_A \subseteq V$ is the set of activity variables. All variables in V_A have Boolean domains.
- D are the domains of the variables in V.
- C is the set of constraints.
- M is a mapping from $(V \setminus V_A) \cup C$ to V_A. We say that v is linked to a if $M(v) = a$. In Example 2, we have $M(v) = a$, $M(v_1) = M(c_1) = a_1$ and $M(v_2) = M(c_2) = a_2$.

An *assignment* to an ACSP is an assignment of values to variables in V. Not all the variables in V are assigned a value, only the *active* variables, defined as follows:

1. Every activity variable is active, hence assigned a value: $True$ or $False$ (we recall that activity variables are always Boolean).
2. For every $v \in V \setminus V_A$, v is assigned a value if and only if $M(v)$ is assigned $True$.

For a constraint $c \in C$, define the *activity set $AS(c)$*:

$$AS(c) := M(c) \cup \bigcup_{\{v \in V \setminus V_A \text{ constraint parameter}\}} M(v)$$

Note that $AS(c)$ is a set of activity variables, (i.e., $AS(c) \subseteq V_A$). Given an assignment, we say a constraint is active if all the activity variables in its activity set are assigned $True$. We say that an assignment is a *solution* if it satisfies all the active constraints.

6.2 Solving ACSPs

Geller and Veksler [13] introduced a solving mechanism based on the notion of *shadow variables*. The mechanism relies on systematic search with a MAC algorithm.

For each constraint $c \in C$ and $v \in V \setminus V_A$ parameter of c, we create a new variable, which is a copy of v. We denote the new variable by $v[AS(c)]$ and say that $v[AS(c)]$ is a shadow of v[3]. We then replace v by $v[AS(c)]$ (i.e., c will operate on $v[AS(c)]$ instead of v). We also create a new propagator that propagates as follows:

- Set $D(v[AS(c)])$ to be $D(v[AS(c)]) \cap D(v)$.
- If the domains of all the activity variables in $AS(c)$ are $\{True\}$ then set $D(v)$ to be equal to $D(v[AS(c)])$.

---

[3] For two constraints c_1, c_2 such that $AS(c_1) = AS(c_2)$, and a variable v which is a parameter of both constraints, it is sufficient to create one shadow variable.

– If $D(v[AS(c)])$ becomes empty; and the domains of all the activity variables in $AS(c)$ are $\{True\}$, except for one variable a, whose domain is $\{True, False\}$: then set the domain of a to $\{False\}$.

The new propagator is invoked during MAC, as long as the domains of all the activity variables in AS contain $True$. If the domain of one of these variables becomes $\{False\}$, the propagator is disabled, and the shadow $v[AS(c)]$ is ignored for the rest of the execution.

Shadow variables allow the solver to run all the possible activity configurations in parallel, before knowing which configuration will eventually end up in the solution. This has the risk of deteriorating run time, since we add many variables and constraints to the problem, and perform propagations that will eventually turn out to have originated in an inactive constraint. However, as shown in [13], various optimizations can reduce the number of shadows. Moreover, the ACSP can be enhanced with smart propagators over activity variables, which make the early propagation even more powerful.

7   Modeling Graph-Based CSPs as Activity CSPs

Given a $GCSP\langle V, D, C, G\rangle$, we create an $ACSP\langle V', V_A, D', C', M\rangle$: for every graph entity t, we create an activity variable, denoted $ac(t)$. We obtain the following ACSP:

– $V_a = \{ac(t) : t \text{ entity in } G\}$.
– $V' = V \cup V_A$.
– $D'(v) = D(v)$ for $v \in V$; $D'(a) = \{True, False\}$ for $a \in V_A$, except for $D'(ac(source))$ and $D'(ac(target))$, which are $\{True\}$.
– $C' = C \cup P$, where the set P of new constraints is defined below.
– For $v \in V$ linked to graph entity t, we have $M(v) = ac(t)$. Same for $c \in C$.

It remains to define P — a new set of constraints, to ensure that a valid path is selected (we denote this set by the letter P since it is the first letter in the word "path").

– For every edge e from n_1 to n_2, we add a constraint $ac(n_1) \wedge ac(n_2)$, and link it to $ac(e)$. These two constraints ensure that if an edge is part of a solution, then the two nodes at its ends are also part of the solution.
– For every node n that is not the target, we add a new constraint, which we link to $ac(n)$: exactly one of $ac(e_1), \dots, ac(e_k)$ is $True$, where $e_1, \dots, e_k$ are the outgoing edges of n.
– We add a similar constraint for incoming edges.

Example 3. This is the ACSP obtained from the GCSP depicted in Fig. 1:

– $V_A = \{ac(n_s), ac(n_t), ac(n_1), ac(n_2)\}$
– $V' = \{x, y, z, w, ac(n_s), ac(n_t), ac(n_1), ac(n_2)\}$, where $M(x) = ac(n_s), M(y) = ac(n_1), M(z) = ac(n_t), M(w) = ac(n_2)$

– The initial domains of x, y, z, w are identical to the domains in the GCSP. For the activity variables: $D'(n_s) = D'(n_t) = \{True\}$, $D'(n_1) = D'(n_2) = \{True, False\}$.
– The constraints, where e_{ij} denotes the edge from n_i to n_j:
  • Linked to $ac(e_{s1})$: $y = x + 1$; $ac(n_s) \wedge ac(n_1)$.
  • Linked to $ac(e_{1t})$: $z = y + 2$; $ac(n_1) \wedge ac(n_t)$.
  • Linked to $ac(e_{s2})$: $w = x - 2$; $ac(n_s) \wedge ac(n_2)$.
  • Linked to $ac(e_{2t})$: $z = w$; $ac(n_2) \wedge ac(n_t)$.
  • Linked to $ac(n_s)$: exactly one of $ac(e_{s1}), ac(e_{s2})$ is $True$.
  • Linked to $ac(n_t)$: exactly one of $ac(e_{1t}), ac(e_{2t})$ is $True$.
  • Linked to $ac(n_1)$: $ac(e_{s1}) = True$; $ac(e_{1t}) = True$.
  • Linked to $ac(n_2)$: $ac(e_{s2}) = True$; $ac(e_{2t}) = True$.

8   Results

We augmented the IBM Research Constraint Solver [1] with an interface capable of reading GCSPs and translating them to activity CSPs, as described in Sect. 7. Two general purpose test generators use this solution.

IBM Genesys Professional Edition Test Generator(Genesys-Pro) [3], uses the IBM Research Constraint Solver for instruction generation for pre-silicon verification. Address translation is an essential part of the process of generating both instruction and operand addresses. This technology enhancement serves not only to ensure correct translation, but also enables generating interesting scenarios by adding hard and soft constraints that target corner cases, thus achieving wide and thorough coverage. We model the translation problem as a separate CSP from the main test generation problem, which is also modeled as a CSP. At run-time, the two problems are connected to create one large CSP. This solution was first deployed on IBM z Systems, enabling Genesys-Pro to test the complex virtual memory management. More recently, this solution was deployed for the IBM Power Systems, and it is currently being developed for ARM architecture.

IBM Threadmill Post-Silicon Exerciser (Threadmill) [6] is a leading IBM solution for post-silicon processor verification. Threadmill is a bare-metal application. Once it is loaded to the system, it continuously generates test-cases, executes them, and checks their results, all on the hardware. To achieve optimal utilization of the silicon platform, it is important to minimize the portion of time spent on test-generation to allow more time for the execution of the test. CSP engines cannot be executed effectively on the silicon, because they are generally implemented in software. However, generating interesting address translation scenarios is also necessary for post-silicon verification. Threadmill generates address translation data in advance, and uses this data later for the on-silicon test-generation. Thus the CSP engine is used offline to generate address translation data. The address translation CSP models used by Threadmill are typically smaller than those used by Genesys-Pro, since they exclude some of the architecture mechanisms (e.g., exceptions), thus reducing the graph size.

We conducted experiments using our solver on a 64 bit Linux machine with 228 GBytes of memory and performance of 2.4 Ghz. Table 1 displays the results

for address translation instances coming from five models: PnoEx is the Power model without exceptions, used for post-silicon validation; P is the full Power model, used for pre-silicon verification; ARM is the full ARM model; Zhost and Zguest represent the two stages of z13 models. All data is for a single stage of the translation. The two stages are implemented in a similar manner, but the CSPs are solved separately. The graphs for the two stages are almost identical; however the guest CSP is more complex, because it has many additional constraints, some that stem from the main problem of instruction generation, and others are used for the invocation of the host translation.

To explain the table's columns, we recall the three steps from a GCSP to a solution:

1. Translate the GCSP to an ACSP (Sect. 7). This step adds new activity variables and new constraints, to ensure that the resulting subgraph is a valid path.
2. Create shadow variables and propagators operating on them (Sect. 6).
3. Solve the resulting CSP.

The first four columns provide information about the GCSP: number of nodes, number of edges, number of variables, and number of constraints. The next two columns refer to the generated ACSP, before the creation of shadow variables (see Step 1 above). The solver adds shadow variables and propagators (Step 2). The seventh and eighth columns give the total number of variables and constraints. The run time, displayed in milliseconds in the last column, is the total run time for all three steps, averaged over 100 executions. Run time of ARM is not optimized since the model is still under development.

Table 1. Performance of the IBM Constraint Solver on a selection of GCSPs used for address translation of several processors

Model	GCSP				ACSP w/o shadows		ACSP with shadows		Time (ms)
	nodes	edges	vars	constraints	vars	constraints	vars	constraints	
PnoEx	7	7	42	69	53	91	103	162	7
P9	8	14	92	151	112	197	294	396	12
ARM	8	19	181	264	206	321	663	814	41
Zhost	16	39	273	450	326	574	1051	1330	31
Zguest	16	38	523	822	575	945	1623	2023	37

9   Conclusions

We presented the solution we developed for generating test cases to verify the complex address translation mechanism. The solution involves modeling the address translation algorithm as a graph-based CSP. We showed how GCSP can be implemented efficiently using activity CSP. The GCSP model is specific for each hardware architecture. Once the architecture is modeled, the user can request various scenarios for verification by adding both hard and soft constraints.

We chose to build our solution on the basis of Constraint Programming (CP) mainly because this technology has proven successful for our test generation tools. CP has several advantages in this domain: a rich, high-level constraint language; support for various variable types such as integers, strings, bitvectors and enumerated types; support for user-defined C++ propagators; soft constraints; and random variable and value ordering. Measuring the quality of test case distribution is a subject of study in itself. We have not made such a study in the scope of this work. However, it is our strong feeling that allowing a level of randomness in their generation is critical to finding bugs in places no one thought to define. By not employing dedicated heuristics to improve the solving process, we allow a level of randomness which we believe, contributes to a good distribution over the solution space. This is very important for generating multiple random-constrained test cases for verification. It may be beneficial to explore usage of other technologies for this problem. In particular, the reviewers suggested examining SMT with bitvector theory. To the best of our knowledge, it is more difficult to introduce random solutions into the SMT process, however this seems like an interesting direction for future study.

In all the sections above, we defined how to model the GCSP so as to provide a path from source to target as the solution. GCSP does not contain anything specific for paths, thus can readily support other types of solutions to select a cycle, or a clique, an independent set, or any other type of subgraph that can be defined using constraints on nodes and edges. Due to this generic aspect of GCSP, we believe that additional applications may suggest problems that can be naturally modeled as GCSPs. These are problems in which the CSP itself — variables and constraints — depends on the selected subgraph. We suggest the search for these new applications as a promising direction for future work.

References

1. http://www.research.ibm.com/haifa/dept/vst/csp.shtml
2. ARM architecture reference manual, ARMv8 for ARMv8-A architecture profile
3. Adir, A., Almog, E., Fournier, L., Marcus, E., Rimon, M., Vinov, M., Ziv, A.: Genesys-Pro: innovations in test program generation for functional processor verification. IEEE Des. Test Comput. **21**(2), 84–93 (2004)
4. Adir, A., Emek, R., Katz, Y., Koyfman, A.: DeepTrans - a model-based approach to functional verification of address translation mechanisms. In: Fourth International Workshop on Microprocessor Test and Verification, Common Challenges and Solutions (MTV 2003), pp. 3–6 (2003)
5. Adir, A., Fournier, L., Katz, Y., Koyfman, A.: DeepTrans - extending the model-based approach to functional verification of address translation mechanisms. In: HLDVT, pp. 102–110. IEEE Computer Society (2006)

6. Adir, A., Golubev, M., Landa, S., Nahir, A., Shurek, G., Sokhin, V., Ziv, A.: Threadmill: a post-silicon exerciser for multi-threaded processors. In: DAC, pp. 860–865. ACM (2011)
7. Anderson, T., Dahlin, M., Systems, O.: Principles and Practice. Memory Management, vol. 3. Recursive Books (2015)
8. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. **90**(1–2), 1636–1642 (1997)
9. Cambazard, H., Bourreau, E.: Conception d'une contrainte globale de chemin. In: JNPC, pp. 107–121 (2004)
10. Do, M.B., Kambhampati, S.: Planning as constraint satisfaction: solving the planning-graph by compiling it into CSP. Artif. Intell. **132**(2), 151–182 (2001)
11. Dooms, G., Deville, Y., Dupont, P.E.: CP(Graph): introducing a graph computation domain in constraint programming. In: Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 211–225. Springer, Heidelberg (2005)
12. Fages, J.G.: Exploitation de structures de graphe en programmation par contraintes. Ph.D. thesis, Ecole des Mines de Nantes (2014)
13. Geller, F., Veksler, M.: Assumption-based pruning in conditional CSP. In: Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 241–255. Springer, Heidelberg (2005)
14. Hamid, A.: Surveying the verification landscape. http://electronicdesign.com/eda/surveying-verification-landscape
15. Kamkin, A., Protsenko, A., Tatarnikov, A.: An approach to test program generation based on formal specifications of caching and address translation mechanisms, vol. 27, pp. 125–138 (2015)
16. Kornykhin, E.V.: Generation of test data for verification of caching mechanisms and address translation in microprocessors. Program. Comput. Softw. **36**(1), 28–35 (2010)
17. Mittal, S., Falkenhainer, B.: Dynamic constraint satisfaction problems. In: AAAI, pp. 25–32. AAAI Press (1990)
18. Naveh, Y., Rimon, M., Jaeger, I., Katz, Y., Vinov, M., Marcus, E., Shurek, G.: Constraint-based random stimuli generation for hardware verification. In: IAAI, pp. 1720–1727. AAAI Press (2006)
19. Le Pape, C., Régin, J.-C., Shaw, P.: Robust and parallel solving of a network design problem. In: Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 633–648. Springer, Heidelberg (2002)
20. Romanescu, B.F., Lebeck, A.R., Sorin, D.J.: Specifying and dynamically verifying address translation-aware memory consistency. In: ASPLOS, pp. 323–334. ACM (2010)
21. Sabin, M., Freuder, E.C., Wallace, R.J.: Greater efficiency for conditional constraint satisfaction. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 649–663. Springer, Heidelberg (2003)

Finding Unsatisfiable Cores of a Set of Polynomials Using the Gröbner Basis Algorithm

Xiaojun Sun[1], Irina Ilioaea[2], Priyank Kalla[1(✉)], and Florian Enescu[2]

[1] Electrical and Computer Engineering, University of Utah, Salt Lake City, UT, USA
{xiaojuns,kalla}@ece.utah.edu
[2] Mathematics and Statistics, Georgia State University, Atlanta, GA, USA
iilioaea1@student.gsu.edu, fenescu@gsu.edu

Abstract. Finding small unsatisfiable subformulas (unsat cores) of infeasible propositional SAT problems is an active area of research. Analogous investigations in the polynomial algebra domain are, however, somewhat lacking. This paper investigates an algorithmic approach to identify a small unsatisfiable core of a set of polynomials, where the corresponding polynomial ideal is found to have an empty variety. We show that such a core can be identified by employing extensions of the Buchberger's algorithm. By further analyzing S-polynomial reductions, we identify certain conditions that are helpful in ascertaining whether or not a polynomial from the given generating set is a part of the unsat core. Our algorithm cannot guarantee a minimal unsat core; the paper describes an approach to refine the identified core. Experiments are performed on a variety of instances using a computer-algebra implementation of our algorithm.

1 Introduction

The Boolean Satisfiability Problem (SAT) is formulated as finding solutions satisfying a set of Boolean equations, or to show that no such solutions exist (unsat). Such problems are often represented in Conjunctive Normal Form (CNF), whereby sets of literal-disjunctions (clauses) must be simultaneously satisfied through some variable assignment. When no solutions exist, it is possible to identify a subset C_c of the original set of clauses C such that C_c is unsat too. This set C_c is called the unsat core of the problem. Unsat cores find application in many areas such as artificial intelligence [1], hardware synthesis [2], formal verification [3], among others [4]. A minimal unsat core is a minimally unsatisfiable subformula (MUS) when any of its subsets is satisfiable. Various unsat core extractors are available [5,6] that can identify smaller/minimal unsat cores from very large unsat problems, and this is an active area of research.

The problem has analogous manifestations in the commutative algebra and algebraic geometry domains. Suppose that we are given a set of polynomials

$F = \{f_1, \ldots, f_s\}$ with coefficients from any field. Suppose, further, that the system of polynomial equations $f_1 = f_2 = \cdots = f_s = 0$ has no common zero – i.e. the system of polynomial constraints is infeasible (or unsat). Can we identify a subset $F_c \subseteq F$ such that the set of polynomials of F_c also has no common zeros? In this paper, we devise algorithmic techniques to identify such infeasible or unsat cores F_c of a set of polynomials F.

Contributions: Computational algebraic geometry – particularly the theory and technology of Gröbner bases (GB) [7] – provides a mechanism to answer the polynomial unsat core question. The given set of polynomials F generates an ideal, and the celebrated Hilbert's Nullstellensatz [8] provides all the tools to characterize the zero-set (varieties) of polynomial ideals. Most Nullstellensatz-related polynomial decision questions can be answered using Gröbner bases. This motivates our investigation into extraction of infeasible cores of the set of polynomials F using Buchberger's algorithm [9] for GB computation.

We apply Buchberger's algorithm on the given set F to first deduce whether or not F has common zeros, by generating the unit ideal from the polynomials. We perform book-keeping of specific information generated during the execution of Buchberger's algorithm – such as the critical pairs used, the polynomials used in reduction of the basis, quotients of polynomials generated, etc. Analysis of this data allows us to identify the unsat core $F_c \subset F$. We identify certain conditions that help us ascertain whether a polynomial $f_i \in F_c$ can be discarded from the unsat core. Our approach cannot guarantee a minimal core, so we further present a technique to refine the unsat core. Experiments are performed on a variety of hardware equivalence checking and combinatorics benchmarks. In many instances, our approach delivers a minimally infeasible subset F_c.

Previous Work: GB techniques have been employed to tackle SAT problems. Nullstellensatz based proof systems, such as the polynomial calculus [10], used GB as a means to derive proof refutation, and to derive complexity lower bounds [11]. GB techniques have also been used in SAT formula preprocessing [12] and for clause learning [13]. Increasing interest in solving Boolean problems with algebraic reasoning has led to algorithm and tool developments for Boolean Gröbner basis [14,15]. While investigations of unsat cores have been made for *linear programs* [16,17], to the best of our knowledge they have not been explored for a *system of polynomial constraints* using Gröbner bases.

2   Preliminaries

Let $\mathbb{F}$ be any field, $\overline{\mathbb{F}}$ its algebraic closure, and $R = \mathbb{F}[x_1, \ldots, x_d]$ the polynomial ring in variables $x_1, \ldots, x_d$ with coefficients from $\mathbb{F}$. A monomial in variables $x_1, \ldots, x_d$ is a power product of the form $X = x_1^{e_1} \cdot x_2^{e_2} \cdots x_d^{e_d}$, where $e_i \in \mathbb{Z}_{\geq 0}, i \in \{1, \ldots, d\}$. A *polynomial* $f \in R$ is written as a finite sum of terms $f = c_1 X_1 + c_2 X_2 + \cdots + c_t X_t$. Here $c_1, \ldots, c_t$ are coefficients and $X_1, \ldots, X_t$ are monomials. To systematically manipulate the polynomials, a monomial order $>$ (or a term order) is imposed on the ring — i.e. a total order on the monomials s.t. multiplication with another monomial preserves the ordering.

The monomials of $f = c_1 X_1 + c_2 X_2 + \cdots + c_t X_t$ are ordered w.r.t. to $>$, such that $X_1 > X_2 > \cdots > X_t$. Subject to $>$, $lt(f) = c_1 X_1$, $lm(f) = X_1$, $lc(f) = c_1$, are the *leading term*, *leading monomial* and *leading coefficient* of f, respectively. While our approach works for any permissible term order, in the paper, we consider terms ordered *degree-lexicographically* ($DEGLEX$), where the monomials are ordered according to their descending total degree, with ties broken lexicographically.

Polynomial Reduction: Let f, g be polynomials. If a non-zero term cX of f is divisible by the leading term of g, then we say that f *is reducible to r* modulo g, denoted $f \xrightarrow{g} r$, where $r = f - \frac{cX}{lt(g)} \cdot g$. Similarly, f can be *reduced (divided) w.r.t. a set of polynomials* $F = \{f_1, \ldots, f_s\}$ to obtain a remainder r. This reduction is denoted $f \xrightarrow{F}_+ r$, and the remainder r has the property that no term in r is divisible by the leading term of any polynomial f_i in F. The division algorithm (*e.g.* Algorithm 1.5.1 [7]) records the *data* related to both the remainders and the quotients in the division process. We will utilize this data to identify the core.

Ideals, Varieties and Nullstellensatz: Let $F = \{f_1, \ldots, f_s\}$ denote the given set of polynomials. An *ideal* $J \subseteq R$ generated by polynomials $f_1, \ldots, f_s \in R$ is:

$$J = \langle f_1, \ldots, f_s \rangle = \{\sum_{i=1}^{s} h_i \cdot f_i : \; h_i \in \mathbb{F}[x_1, \ldots, x_d]\}.$$

The polynomials $f_1, \ldots, f_s$ form the *basis* or the *generators* of J.

The set of all solutions to a given system of polynomial equations $f_1 = \cdots = f_s = 0$ is called the *variety*, which depends upon the ideal $J = \langle f_1, \ldots, f_s \rangle$ generated by the polynomials. The variety is denoted by $V(J) = V_{\mathbb{F}}(J) = V_{\mathbb{F}}(f_1, \ldots, f_s)$ and defined as:

$$V_{\mathbb{F}}(J) = V_{\mathbb{F}}(f_1, \ldots, f_s) = \{\mathbf{a} \in \mathbb{F}^d : \forall f \in J, f(\mathbf{a}) = 0\},$$

where $\mathbf{a} = (a_1, \ldots, a_d) \in \mathbb{F}^d$ denotes a point in the affine space.

Theorem 1 (The Weak Nullstellensatz [8]). *Let $J = \langle f_1, \ldots, f_s \rangle$ be an ideal in the ring $\mathbb{F}[x_1, \ldots, x_d]$ and $V_{\overline{\mathbb{F}}}(J)$ be its variety over $\overline{\mathbb{F}}$. Then $V_{\overline{\mathbb{F}}}(J) = \emptyset \iff J = \mathbb{F}[x_1, \ldots, x_d] \iff 1 \in J$.*

The Weak Nullstellensatz provides a mechanism to ascertain whether or not a given system of polynomials has a feasible solution. This is deduced by testing whether the unit element is a member of the ideal J. This *ideal membership test* requires the computation of a Gröbner basis.

Definition 1 (Gröbner Basis [7]). *For a monomial ordering $>$, a set of nonzero polynomials $G = \{g_1, g_2, \cdots, g_t\}$ contained in an ideal J, is called a Gröbner basis for J iff $\forall f \in J$, $f \neq 0$ there exists $i \in \{1, \cdots, t\}$ such that $lm(g_i)$ divides $lm(f)$; i.e., $G = GB(J) \Leftrightarrow \forall f \in J : f \neq 0, \exists g_i \in G : lm(g_i) \mid lm(f)$. Equivalently, $G = \{g_1, g_2, \cdots, g_t\}$ is a Gröbner basis for J iff division by G of any polynomial in J gives the remainder 0; i.e. $G = GB(J) \iff \forall f \in J, f \xrightarrow{g_1, \ldots, g_t}_+ 0$.*

Buchberger's algorithm [9], shown in Algorithm 1, takes as input the given polynomials $F = \{f_1, \ldots, f_s\}$, and computes the Gröbner basis $G = \{g_1, \ldots, g_t\}$. The algorithm takes pairs of polynomials (f_i, f_j), and computes their S-polynomial $(Spoly(f_i, f_j))$:

$$Spoly(f_i, f_j) = \frac{L}{lt(f_i)} \cdot f_i - \frac{L}{lt(f_j)} \cdot f_j$$

where $L = \mathrm{LCM}(lm(f_i), lm(f_j))$. $Spoly(f_i, f_j)$ cancels the leading terms of f_i and f_j. The computation $Spoly(f_i, f_j) \xrightarrow{G'}_+ g_{ij}$ results in a remainder g_{ij}, which if non-zero, provides an element with new leading term in the generating set. The algorithm terminates when for all pairs (f_i, f_j), $Spoly(f_i, f_j) \xrightarrow{G'}_+ 0$.

Algorithm 1. Buchberger's Algorithm

Input: $F = \{f_1, \ldots, f_s\}$
Output: $G = \{g_1, \ldots, g_t\}$
$G := F$;
repeat
    $G' := G$;
    for *each pair $\{f_i, f_j\}, i \neq j$ in G'* **do**
        $Spoly(f_i, f_j) \xrightarrow{G'}_+ g_{ij}$;
        if $g_{ij} \neq 0$ **then**
            $G := G \cup \{g_{ij}\}$;
        end
    end
until $G = G'$;

A Gröbner basis G may contain redundant elements. To remove these redundant elements, G is first made *minimal* and subsequently *reduced.* Subject to the given term order $>$, the reduced GB $G_r = \{g_1, \ldots, g_t\}$ is a unique, canonical representation of ideal J. In the context of Nullstellensatz, $V_{\bar{\mathbb{F}}}(J) = \emptyset \iff G_r = GB(J) = \{1\}$. This implies that for ideals with empty variety, there exists a sequence of $Spoly(f_i, f_j)$ reductions in the reduced GB computation that leads to the unit element. We now show that by analyzing this data, the unsat core F_c of the given generating set $F = \{f_1, \ldots, f_s\}$ can be identified.

3   Motivating the Search for an Unsat Core

Problem 1. *Let $F = \{f_1, \ldots f_s\}$ be a set of multivariate polynomials in the ring $R = \mathbb{F}[x_1, \ldots, x_d]$ that generate ideal $J = \langle f_1, \ldots, f_s \rangle \subset R$. Suppose that it is known that $V(J) = \emptyset$, or it is determined to be so by applying the Gröbner basis algorithm. Identify a subset of polynomials $F_c \subseteq F, J_c = \langle F_c \rangle$, such that $V(J_c) = \emptyset$ too. We call F_c the infeasible core or the unsat core of F.*

It is not hard to motivate that an unsat core should be identifiable using the Gröbner basis algorithm: Assume that $F_c = F - \{f_j\}$. If $GB(F) = GB(F_c) = \{1\}$, then it implies that f_j is a member of the ideal generated by $(F - \{f_j\})$, i.e. $f_j \in \langle F - \{f_j\} \rangle$. Thus f_j can be composed of the other polynomials of F_c, so f_j is not a part of the unsat core, and it can be safely discarded from F_c. This can be identified by means of the GB algorithm for this ideal membership test.

A näive way (and inefficient way) to identify *a minimal core* using the GB computation is as follows: select a polynomial f_i and see if $V(F_c - \{f_i\}) = \emptyset$ (i.e. if reduced $GB(F_c - \{f_i\}) = \{1\}$). If so, discard f_i from the core; otherwise retain f_i in F_c. Select a different f_i and continue until all polynomials f_i are visited for inclusion in F_c. This approach will produce a minimal core, as we would have tested each polynomial f_i for inclusion in the core. This requires $O(|F|)$ calls to the GB engine, which is really impractical.

3.1   The Refutation Tree of the GB Algorithm: Find F_c from F

We investigate if it is possible to identify a core by analyzing the $Spoly(f_i, f_j) \xrightarrow{F}_{+} g_{ij}$ reductions in Buchberger's algorithm. Since F is itself an unsat core, definitely *there exists a sequence of Spoly reductions in Buchberger's algorithm where $Spoly(f_i, f_j) \xrightarrow{F}_{+} 1$ is achieved.* Moreover, polynomial reduction algorithms can be suitably modified to record which polynomials from F are used in the division leading to $Spoly(f_i, f_j) \xrightarrow{F}_{+} 1$. This suggests that we should be able to identify a core by recording the *data* generated by Buchberger's algorithm — namely, the critical pairs (f_i, f_j) used in the Spoly computations, and the polynomials from F used to cancel terms in the reduction $Spoly(f_i, f_j) \xrightarrow{F}_{+} 1$. The following example motivates our approach to identify $F_c \subseteq F$ using this data:

Example 1. Consider the following set of polynomials $F = \{f_1, \ldots, f_9\}$:

$f_1 : abc + ab + ac + bc$	$f_5 : bc + c$
$\quad + a + b + c + 1$	$f_6 : abd + ad + bd + d$
$f_2 : b$	$f_7 : cd$
$f_3 : ac$	$f_8 : abd + ab + ad + bd + a + b + d + 1$
$f_4 : ac + a$	$f_9 : abd + ab + bd + b$

Assume $>_{DEGLEX}$ monomial ordering with $a > b > c > d$. Let $F = \{f_1, \ldots, f_9\}$ and $J = \langle F \rangle \subset \mathbb{F}_2[a, b, c, d]$ where $\mathbb{F}_2 = \{0, 1\}$ is the finite field of 2 elements. Then $V(J) = \emptyset$ as $GB(J) = 1$. The set F consists of 4 *minimal* cores: $F_{c1} = \{f_1, f_2, f_3, f_4, f_7, f_8\}, F_{c2} = \{f_2, f_4, f_5, f_6, f_8\}, F_{c3} = \{f_2, f_3, f_4, f_6, f_8\}$, and $F_{c4} = \{f_1, f_2, f_4, f_5\}$.

Buchberger's algorithm terminates to a unique reduced GB, irrespective of the order in which the critical pairs (f_i, f_j) are selected and reduced by operation $Spoly(f_i, f_j) \xrightarrow{F}_{+} g_{ij}$. Let us suppose that in the GB computation corresponding

to Example 1, the first 3 critical *Spoly* pairs analyzed are $(f_1, f_2), (f_3, f_4)$ and (f_2, f_5). It turns out that the Spoly-reductions corresponding to these 3 pairs lead to the unit ideal. Recording the data corresponding to this sequence of reductions is depicted by means of a graph in Fig. 1. We call this graph a *refutation tree*.
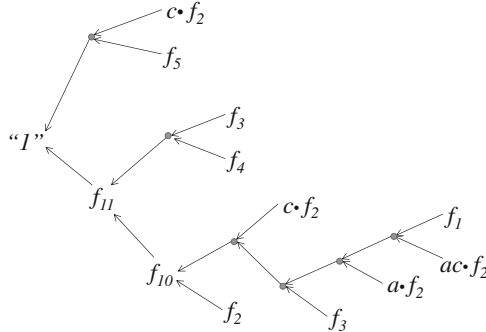
Fig. 1. Generating refutation trees to record unsat cores.

In the figure, the nodes of the graph correspond to the polynomials utilized in Buchberger's algorithm. The leaf nodes always correspond to polynomials from the given generating set. An edge e_{ij} from node i to node j signifies that the polynomial at node j results from polynomial at node i. For example, consider the computation $Spoly(f_1, f_2) \xrightarrow{F}_+ f_{10}$, where $f_{10} = a + c + 1$. Since $Spoly(f_1, f_2) = f_1 - ac \cdot f_2$, the leaves corresponding to f_1 and $ac \cdot f_2$ are created. The reduction $Spoly(f_1, f_2) \xrightarrow{F}_+ f_{10}$ is carried out as the following sequence of 1-step divisions: $Spoly(f_1, f_2) \xrightarrow{a \cdot f_2} \xrightarrow{f_3} \xrightarrow{c \cdot f_2} \xrightarrow{f_2} f_{10}$. This is depicted as the bottom subtree in the figure, terminating at polynomial f_{10}. Moreover, the multiplication $a \cdot f_2$ implies that division by f_2 resulted in the quotient a. The refutation tree of Fig. 1 shows further that $Spoly(f_3, f_4) \xrightarrow{f_{10}} f_{11} = c + 1$ and, finally, $Spoly(f_5, f_2) \xrightarrow{f_{11}} 1$.

To identify an $F_c \subset F$, we start from the refutation node "1", and traverse the graph in reverse, all the way up to the leaves. Then, all the leaves in the transitive fanin of "1" constitute an unsat core. The polynomials (nodes) that do not lie in the transitive fanin of "1" can be safely discarded from F_c. From Fig. 1, $F_c = \{f_1, f_2, f_3, f_4, f_5\}$ is identified as an unsat core of F.

4   Reducing the Size of the Infeasible Core F_c

The core F_c obtained from the aforementioned procedure may contain redundant elements which could be discarded. For example, consider the core $F_c = \{f_1, \ldots, f_5\}$ generated in the previous section. While F_c is a smaller infeasible core of F, it is not minimal. In fact, Example 1 shows that $F_{c4} = \{f_1, f_2, f_4, f_5\}$

is the minimal core, where $F_{c4} \subset F_c$. Clearly, the polynomial $f_3 \in F_c$ is a redundant element of the core and can be discarded. We will now describe techniques to further reduce the size of the unsat core by identifying such redundant elements. For this purpose, we will have to perform a more systematic book-keeping of the data generated during the execution of Buchberger's algorithm and the refutation tree.

4.1 Identifying Redundant Polynomials from the Refutation Tree

We record the S-polynomial reduction $Spoly(f_i, f_j) \xrightarrow{F}_+ g_{ij}$ that give a non-zero remainder when divided by the system of polynomials F at that moment. The remainder g_{ij} is a polynomial combination of $Spoly(f_i, f_j)$ and the current basis F; thus, it can be represented as:

$$g_{ij} = S(f_i, f_j) + \sum_{k=1}^{m} c_k f_k, \tag{1}$$

where $0 \neq c_k \in \mathbb{F}[x_1, \ldots, x_d]$ and $\{f_1, \ldots, f_m\}$ is the "current" system of polynomials F. For each non-zero g_{ij}, we will record the following data:

$$((g_{ij})(f_i, h_{ij})(f_j, h_{ji})|(c_{k1}, f_{k1}), (c_{k2}, f_{k2}), \ldots, (c_{kl}, f_{kl})) \tag{2}$$

In Eqs. (1) and (2), g_{ij} denotes the remainder of the S-polynomial $Spoly(f_i, f_j)$ modulo the current system of polynomials $f_1, \ldots, f_m$, and we denote by

$$h_{ij} := \frac{\text{LCM}(lm(f_i), lm(f_j))}{lt(f_i)}, h_{ji} := -\frac{\text{LCM}(lm(f_i), lm(f_j))}{lt(f_j)}$$

the coefficients of f_i, respectively f_j, in the S-polynomial $Spoly(f_i, f_j)$. Furthermore, in Eq. (2), $(c_{k1}, \ldots c_{kl})$ are the respective quotients of division by polynomials $(f_{k1}, \ldots, f_{kl})$, generated during the $Spoly$ reduction.

Example 2. Revisiting Example 1, and Fig. 1, the data corresponding to $Spoly(f_1, f_2) \xrightarrow{F}_+ g_{12} = f_{10}$ reduction is obtained as the following sequence of computations:

$$f_{10} = g_{12} = f_1 - acf_2 - af_2 - f_3 - cf_2 - f_2.$$

As the coefficient field is $\mathbb{F}_2$ in this example, $-1 = +1$, so:

$$f_{10} = g_{12} = f_1 + acf_2 + af_2 + f_3 + cf_2 + f_2$$

is obtained. The data is recorded according to Eq. (2):

$$((f_{10} = g_{12}), (f_1, 1)(f_2, ac)|(a, f_2), (1, f_3), (c, f_2), (1, f_2))$$

Our approach and the book-keeping terminates when we obtain "1" as the remainder of some S-polynomial modulo the current system of generators. As an output of the Buchberger's algorithm, we can obtain not only the Gröbner basis $G = \{g_1, \ldots, g_t\}$, but also a matrix M of polynomials such that:

$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_t \end{bmatrix} = M \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \end{bmatrix} \tag{3}$$

Each element g_i of G is a polynomial combination of $\{f_1, \ldots, f_s\}$. Moreover, this matrix M is constructed precisely using the data that is recorded in the form of Eq. (2). We now give a condition when the matrix M may identify some redundant elements.

Theorem 2. *With the notations above, we have that a core for the system of generators $F = \{f_1, \ldots, f_s\}$ of the ideal J is given by the union of those f_i's from F that appear in the data recorded above and correspond to the nonzero entries in the matrix M.*

Proof. In our case, since the variety is empty, and hence the ideal is unit, we have that $G = \{g_1 = 1\}$ and $t = 1$. Therefore $M = [a_1, \ldots, a_s]$ is a vector. Then the output of the algorithm gives: $1 = a_1 f_1 + \cdots + a_s f_s$. Clearly, if $a_i = 0$ for some i then f_i does not appear in this equation and should not be included in the infeasible core of F.

Example 3. Corresponding to Example 1 and the refutation tree shown in Fig. 1, we discover that the polynomial f_3 is used only twice in the division process. In both occasions, the quotient of the division is 1. From Fig. 1, it follows that:

$$1 = (f_2 + f_5) + \cdots + \mathbf{1} \cdot \mathbf{f_3} + \cdots + \mathbf{1} \cdot \mathbf{f_3} + \cdots + (f_1 + f_2) \tag{4}$$

Since $1 + 1 = 0$ over $\mathbb{F}_2$, we have that the entry in M corresponding to f_3 is 0, and so f_3 can be discarded from the core.

4.2   The GB-Core Algorithm Outline

The following steps describe an algorithm (GB-Core) that allows us to compute a refutation tree of the polynomial set and corresponding matrix M.

Inputs: Given a system of polynomials $F = \{f_1, \ldots, f_s\}$, a monomial order $>$ on $\mathbb{F}[x_1, \ldots, x_d]$.

S-polynomial Reduction: We start computing the S-polynomials of the system of generators $\{f_1, \ldots, f_s\}$, then divide each of them by the current basis $G = \{f_1, \ldots, f_s, \ldots, f_m\}$, which is the intermediate result of Buchberger's algorithm. In this way, we obtain expressions of the following type:

$$g_{ij} = \underbrace{h_{ij} f_i + h_{ji} f_j}_{Spoly(f_i, f_j)} + \sum_{k=1}^{m} c_k f_k \tag{5}$$

If the remainder g_{ij} is non-zero, we denote it by f_{m+1} and add it to the current set of generators G. We also record the data as in Eq. (2):

$$((f_{m+1} = g_{ij})(f_i, h_{ij})(f_j, h_{ji})|(c_{k1}, f_{k1}), (c_{k2}, f_{k2}), \ldots, (c_{kl}, f_{kl}))$$

This data forms a part of the refutation tree rooted at node f_{m+1}.

Recording the Coefficients: In Eq. (5) we obtain a vector of polynomial coefficients c_k where $k > s$. These coefficients are associated with new elements (remainders) in the Gröbner basis, that are not a part of the unsat core. Since each polynomial f_k, $(k > s)$ is generated by $\{f_1, \ldots, f_s\}$, we can re-express f_k in terms of $\{f_1, \ldots, f_s\}$. Thus, each $f_k, k > s$ can be written as $f_k = d_1 f_1 + \cdots + d_s f_s$. This process adds a new row $(d_1, \ldots, d_s)$ to the coefficient matrix M.

Termination and Refutation Tree Construction: We perform S-polynomial reductions and record these coefficients generated during the division until the remainder $f_m = 1$ is encountered. The corresponding data is stored in a data-structure D corresponding to Eq. (2). The matrix M is also constructed. From this recorded data the refutation tree can be easily derived.

We start with the refutation node "$f_m = 1$":

$$((f_m = 1)(f_i, h_{ij})(f_j, h_{ji})|(c_{k1}, f_{k1}), (c_{k2}, f_{k2}), \ldots, (c_{kl}, f_{kl}))$$

and recursively substitute the expressions for the polynomials f_k $(k > s)$ until we obtain the tree with all the leaf nodes corresponding to the original set of polynomials $\{f_1, \ldots, f_s\}$. Algorithm 2 describes this data recording through which the refutation tree T and the matrix M is derived.

Algorithm 2. GB-core algorithm (based on Buchberger's algorithm)

Input: $F = \{f_1, \ldots, f_s\} \in \mathbb{F}[x_1, \ldots, x_d], f_i \neq 0$
Output: Refutation tree T and coefficients matrix M
1: Initialize: list $G \leftarrow F$; Dataset $D \leftarrow \emptyset$; $M \leftarrow s \times s$ unit matrix
2: **for** each pair $(f_i, f_j) \in G$ **do**
3:     $f_{sp}, (f_i, h_{ij})(f_j, h_{ji}) \leftarrow \text{Spoly}(f_i, f_j)$ {f_{sp} is the S-polynomial}
4:     $g_{ij}|(c_{k1}, f_{k1}), \ldots, (c_{kl}, f_{kl}) \leftarrow (f_{sp} \xrightarrow{G}_+ g_{ij})$
5:     **if** $g_{ij} \neq 0$ **then**
6:         $G \leftarrow G \cup g_{ij}$
7:         $D \leftarrow D \cup ((g_{ij})(f_i, h_{ij})(f_j, h_{ji})|(c_{k1}, f_{k1}), (c_{k2}, f_{k2}), \ldots, (c_{kl}, f_{kl}))$
8:         Update matrix M
9:     **end if**
10:    **if** $g_{ij} = 1$ **then**
11:        Construct T from D
12:        Return(T, M)
13:    **end if**
14: **end for**

Notice that the core can actually be derived directly from the matrix M. However, we also construct the refutation tree T as it facilitates an iterative refinement of the core, which is described in the next section.

5   Iterative Refinement of the Unsat Core

As with most other unsat core extractors, our algorithm also cannot generate a minimal core in one execution. To obtain a smaller core, we re-execute our algorithm with the core obtained in the current iteration. We describe two heuristics that are applied to our algorithm to increase the likelihood of generating a smaller core in the next iteration.

An effective heuristic should increase the chances that the refutation "1" is composed of fewer polynomials. In our GB-core algorithm, we use a strategy to pick critical pairs such that polynomials with larger indexes get paired *later* in the order:

$$(f_1, f_2) \rightarrow (f_1, f_3) \rightarrow (f_2, f_3) \rightarrow (f_1, f_4) \rightarrow (f_2, f_4) \rightarrow \cdots$$

Moreover, for the reduction process $Spoly(f_i, f_j) \xrightarrow{F}_+ g_{ij}$, we pick divisor polynomials from F following the increasing order of polynomial indexes. Therefore, by relabeling the polynomial indexes, we can affect their chances of being selected in the unsat core. We use two criteria to affect the polynomial selection in the unsat core. One corresponds to the *refutation distance*, whereas the other corresponds to the *frequency* with which a polynomial appears in the refutation tree.

Definition 2 (Refutation Distance). *Refutation distance of a polynomial f_i in a refutation tree corresponds to the number of edges on the shortest path from refutation node "1" to any leaf node that represents polynomial f_i.*

On a given refutation tree, polynomials with shorter refutation distances are used as divisors in later stages of polynomial reductions; which implies that they may generally have lower-degree leading terms. This is because we impose a degree-lexicographic term order, and successive divisions (term cancellations) reduce the degree of the remainders. However, what is more desirable is to use these polynomials with lower-degree leading terms earlier in the reduction, as they can cancel more terms. This may prohibit other (higher-degree) polynomials from being present in the unsat core.

Similarly, the motivation for using the *frequency of occurrence* of f_i in the refutation tree is as follows: polynomials that appear frequently in the refutation tree may imply that they have certain properties (leading terms) that give them a higher likelihood of being present in the unsat core.

We apply both heuristics: after the first iteration of the GB-core algorithm, we analyze the refutation tree T and sort the polynomials in the core by the refutation distance criterion, and use the frequency criterion as the tie-breaker. The following example illustrates our heuristic.

Example 4. Consider a set of 6 polynomials over $\mathbb{F}_2$ of an infeasible instance.

$$f_1 : x_1x_3 + x_3; \quad f_2 : x_2 + 1$$
$$f_3 : x_2x_3 + x_2; \quad f_4 : x_2x_3$$
$$f_5 : x_2x_3 + x_2 + x_3 + 1; \quad f_6 : x_1x_2x_3 + x_1x_3$$

After the first iteration of the GB-core algorithm, the core is identified as $\{f_1, f_2, f_3, f_4\}$, and we obtain a refutation tree as shown in Fig. 2(a).
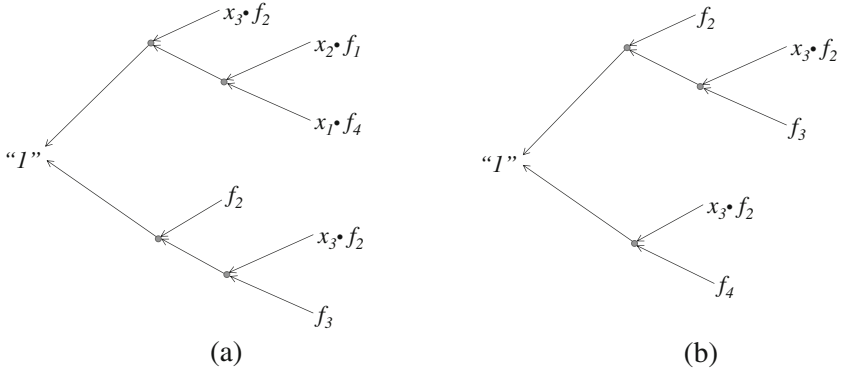
Fig. 2. Refutation trees of core refinement example

The refutation distance corresponding to polynomial f_2 is equal to 2 levels. Note that while three leaf nodes in Fig. 2(a) correspond to f_2, the shortest distance from "1" to any f_2 node is 2 levels. The refutation distance and frequency measures of other polynomials are identical – equal to 3 and 1, respectively – so their relative ordering is unchanged. We reorder f_2 to be the polynomial with the smallest index. We re-index the polynomial set $f_1' = f_2, f_2' = f_1, f_3' = f_3, f_4' = f_4$ and apply our GB-core algorithm on the core $\{f_1', f_2', f_3', f_4'\}$. The result is shown in Fig. 2(b) with the core identified as $\{f_1', f_3', f_4'\} = \{f_2, f_3, f_4\}$. Further iterations do not refine the core – i.e. a fix point is reached.

6   Refining the Unsat Core Using Syzygies

The unsat core obtained through our GB-core algorithm is by nature a refutation polynomial that equals to 1, i.e. $1 = \sum_{i=1}^{s} c_i \cdot f_i$, where $0 \neq c_i \in \mathbb{F}[x_1, \ldots, x_d]$ and the polynomials $F = \{f_1, \ldots, f_s\}$ form a core. Suppose that a polynomial $f_k \in F$ can be represented using a combination of the rest of the polynomials of the core, e.g.: $f_k = \sum_{j \neq k} c_j' f_j$. Then we can substitute f_k in terms of the other polynomials in the refutation. Thus, f_k is redundant and can be dropped from the core. One limitations of the GB-core algorithm and the re-labeling/refinement

strategy is that they cannot easily identify such polynomials. We present an approach targeted to identify such combinations to further refine the core.

During the execution of Buchberger's algorithm, many critical pairs (f_i, f_j) do not add any new polynomials in the basis when $Spoly(f_i, f_j) \xrightarrow{F}_+ 0$ gives zero remainder. Naturally, for the purpose of the GB computation, this data is discarded. However, our objective is to gather more information from each GB iteration so as to refine the core. Therefore, we further record the quotient-divisor data from S-polynomial reductions that result in the remainder 0. Every $Spoly(f_i, f_j) \xrightarrow{F}_+ 0$ implies that some polynomial combination of $\{f_1, \ldots, f_s\}$ vanishes: i.e. $c_1 f_1 + c_2 f_2 + \cdots + c_s f_s = 0$, for some $c_1, \ldots, c_s$. These elements $(c_1, \ldots, c_s)$ form a syzygy on $f_1, \ldots, f_s$.

Definition 3 (Syzygy [8]). *Let $F = \{f_1, \ldots, f_s\}$. A syzygy on $f_1, \ldots, f_s$ is an s-tuple of polynomials $(c_1, \ldots, c_s) \in (\mathbb{F}[x_1, \ldots, x_d])^s$ such that $\sum_{i=1}^{s} c_i \cdot f_i = 0$.*

For each $Spoly(f_i, f_j) \xrightarrow{F}_+ 0$ reduction, we record the information on corresponding syzygies as in Eq. (6), also represented in matrix form in Eq. (7):

$$\begin{cases} c_1^1 f_1 + c_2^1 f_2 + \cdots + c_s^1 f_s = 0 \\ c_1^2 f_1 + c_2^2 f_2 + \cdots + c_s^2 f_s = 0 \\ \vdots \\ c_1^m f_1 + c_2^m f_2 + \cdots + c_s^m f_s = 0 \end{cases} \quad (6) \qquad \begin{bmatrix} c_1^1 & c_2^1 & \cdots & c_s^1 \\ c_1^2 & c_2^2 & \cdots & c_s^2 \\ \vdots & \vdots & \ddots & \vdots \\ c_1^m & c_2^m & \cdots & c_s^m \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \end{bmatrix} = 0 \quad (7)$$

Here $\{f_1, f_2, \ldots, f_s\}$ is the given core. Take one column of the syzygy matrix (e.g. the set of polynomials in j-th column $c_j^1, c_j^2, \ldots, c_j^m$) and compute its reduced Gröbner basis G_r. If $G_r = \{1\}$, then it means that there exists some polynomial vector $[r_1, r_2, \ldots, r_m]$ such that $1 = r_1 c_j^1 + r_2 c_j^2 + \cdots + r_m c_j^m = \sum_{i=1}^{m} r_i c_j^i$. If we multiply each row i in the matrix of Eq. (7) with r_i, and sum up all the rows, we will obtain the following equation:

$$\begin{bmatrix} \sum_{i=1}^{m} r_i c_1^i & \cdots & 1 & \cdots & \sum_{i=1}^{m} r_i c_s^i \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \end{bmatrix} = 0 \quad (8)$$

This implies that

$$\sum_{i=1}^{m} r_i c_1^i f_1 + \cdots + f_j + \cdots + \sum_{i=1}^{m} r_i c_s^i f_s = 0,$$

or that f_j is a polynomial combination of $f_1, \ldots, f_s$ (excluding f_j). Subsequently, we can deduce that f_j can be discarded from the core. By repeating this procedure, some redundant polynomials can be identified and size of unsat core can be reduced further.

Example 5. Revisiting Example 1, execute the GB-core algorithm and record the syzygies on $f_1, \ldots, f_s$ corresponding to the S-polynomials that give 0 remainder. The coefficients can be represented as entries in matrix shown below. For example, the first row in the matrix corresponds to the syzygies generated by $Spoly(f_1, f_3) \xrightarrow{F}_+ 0$.

$$
\begin{array}{c}
\\
Spoly(f_1, f_3) \\
Spoly(f_2, f_3) \\
Spoly(f_1, f_4) \\
Spoly(f_2, f_4) \\
Spoly(f_1, f_5)
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 & f_9 & f_{10} \end{array} \\
\left(\begin{array}{cccccccccc}
1 & a+c+1 & b+1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & ac & b & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & c+1 & 1 & b & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & ac+a & 0 & b & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & a+c+1 & 0 & 0 & a & 0 & 0 & 0 & 0 & 1
\end{array}\right)
\end{array} \quad (9)
$$

Usually, we need to generate extra columns compared to the syzygy matrix of Eq. (7). In this example, we need to add an extra column for the coefficient of f_{10}. This is because f_{10} is not among the original generating set; however, some S-polynomial pairs require this new remainder f_{10} as a divisor during reduction. In order to remove this extra column, we need to turn the non-zero entries in this column to 0 through standard matrix manipulations.

Recall that we record f_{10} in M as a nonzero remainder when reducing S-polynomial pair $Spoly(f_1, f_2) \xrightarrow{F}_+ f_{10}$. We extract this information from the coefficient matrix M:

$$(1 \quad ac+a+c+1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0)$$

It represents f_{10} is a combination of f_1 to f_9:

$$f_{10} = f_1 + (ac+a+c+1)f_2 + f_3$$

It can be written in the same syzygy matrix form (with column f_{10} present) as follows:

$$
\begin{array}{c}
\\
Spoly(f_1, f_2)
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 & f_9 & f_{10} \end{array} \\
\left(\begin{array}{cccccccccc}
1 & ac+a+c+1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
\end{array} \quad (10)
$$

By adding this row vector (Eq. (10)) to the rows in Eq. (9) corresponding to the non-zero entries in the column for f_{10}, we obtain the syzygy matrix only for the polynomials in the core:

$$
\begin{array}{c}
\\
Spoly(f_1, f_3) \\
Spoly(f_2, f_3) \\
Spoly(f_1, f_4) \\
Spoly(f_2, f_4) \\
Spoly(f_1, f_5)
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 & f_9 \end{array} \\
\left(\begin{array}{ccccccccc}
0 & ac & b & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & ac & b & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & ac+a & 0 & b & 0 & 0 & 0 & 0 & 0 \\
0 & ac+a & 0 & b & 0 & 0 & 0 & 0 & 0 \\
0 & ac & 1 & 0 & a & 0 & 0 & 0 & 0
\end{array}\right)
\end{array}
$$

There is a "1" entry in the f_3 column. The last row implies that f_3 is a combination of f_2, f_5 ($f_3 = acf_2 + af_5$), so f_3 can be discarded from the core.

While the syzygy heuristic gathers extra information from the GB computation, it is still not sufficient to derive all polynomial dependencies. In Buchberger's algorithm, many S-polynomials reduce to zero, so the number of rows of the syzygy matrix can be much larger than the size of original generating set. Full GB computation on each column of the syzygy matrix can become prohibitive to apply iteratively. For this reason, we only apply the syzygy heuristic on the smaller reduced core given by our iterative refinement algorithm.

Overall Approach for Unsat Core Extraction: (i) Given the set $F = \{f_1, \ldots, f_s\}$, we apply the GB-core algorithm, record the data D, M (Sect. 4) and the syzygies S on $f_1, \ldots, f_s$. (ii) From M, we obtain a core $F_c \subseteq F$. (iii) Iteratively refine F_c (Sect. 5) until $|F_c|$ cannot be reduced further. (iv) Apply the syzygy-heuristic (Sect. 6) to identify if some $f_k \in F_c$ is a combination of other polynomials in F_c; all such f_k are discarded from F_c. This gives us the final unsat core F_c.

7 Experiment Results

We have implemented our core extraction approach (the GB-Core and the refinement algorithms) using the SINGULAR symbolic algebra computation system [v. 3-1-6] [18]. With our tool implementation, we have performed experiments to extract a minimal unsat core from a given set of polynomials. Our experiments run on a desktop with 3.5 GHz Intel Core^{TM} i7–4770 K Quad-core CPU, 16 GB RAM and 64-bit Ubuntu Linux OS. The experiments are shown in Table 1.

Gröbner basis is not an efficient engine for solving contemporary industry-size CNF-SAT benchmarks, as the translation from CNF introduces too many variables and clauses for GB engines to handle. In order to validate our approach, we use a somewhat customized benchmark library: (i) "aim-100" is a modified version of the random 3-SAT benchmark "aim-50/100", modified by adding some redundant clauses; (ii) The "subset" series are generated for random subset sum problems; (iii) "cocktail" is similarly revised from a combination of factorization and a random 3-SAT benchmark; (iv) and "phole4/5" are generated by adding redundant clauses to pigeon hole benchmarks; (v) Moreover, SMPO and RH benchmarks correspond to hardware equivalence checking instances of sequential Galois field normal basis modulo multiplier circuits [19,20], compared against a golden model *spec*. Similarly, the "MasVMon" benchmarks are the equivalence checking circuits corresponding to Mastrovito multipliers compared against Montgomery multipliers [21]. Some of these are available as CNF formulae, whereas others were available directly as polynomials over finite fields. The CNF formulae are translated as polynomial constraints over $\mathbb{F}_2$ (as shown in [12]), and the GB-Core algorithm and the refinement approach is applied.

In Table 1, #Polys denotes the given number of polynomials from which a core is to be extracted. #MUS is the *minimal* core either extracted by PicoMUS (for CNF benchmarks) or exhaustive deletion method (for non-CNF bencmarks). #GB-core iterations corresponds to the number of calls to the GB-core engine to

Table 1. Results of running benchmarks using our tool. Asterisk(*) denotes that the benchmark was not translated from CNF. Our tool comprises 3 parts: part I runs a single GB-core algorithm, part II applies the iterative refinement heuristic to run the GB-core algorithm iteratively, part III applies the syzygy heuristic.

Benchmark	# Polys	# MUS	Size of core			# GB-core iterations	Runtime (sec)			Runtime of PicoMUS (sec)
			I	II	III		I	II	III	
5 × 5 SMPO	240	137	169	137	137	8	1222	1938	1698	< 0.1
4 × 4 SMPO*	84	21	21	21	21	1	125	0.3	29	-
3 × 3 SMPO*	45	15	15	15	15	1	6.6	0.2	5.7	-
3 × 3 SMPO	17	2	2	2	2	1	0.07	0.01	0.01	< 0.1
4 × 4 MasVMont*	148	83	83	83	83	1	23	139	12	-
3 × 3 MasVMont*	84	53	53	53	53	1	4.3	4.6	0.9	-
2 × 2 MasVMont	27	23	24	23	23	2	1.3	1.0	80	< 0.1
5 × 5 RH*	142	34	48	35	35	4	997	1.0	80	-
4 × 4 RH*	104	35	43	36	36	3	96	5.7	0.6	-
3 × 3 RH*	50	20	20	20	20	1	2.9	3.5	10	-
aim-100	79	22	22	22	22	1	43	0.7	0.2	< 0.1
cocktail	135	4	6	4	4	2	51	0.01	0.01	< 0.1
subset-1	100	6	6	6	6	1	2.4	0.01	0.01	< 0.1
subset-2	141	19	37	23	21	2	12	1.6	1.1	< 0.1
subset-3	118	16	13	12	11	2	8.6	0.2	0.07	< 0.1
phole4	104	10	16	16	10	1	4.3	0.2	0.5	< 0.1
phole5	169	19	30	25	19	3	12	3.2	2.7	< 0.1

arrive at the reduced unsat core. The second last column shows the improvement in the minimal core size by applying the syzygy heuristic on those cases which cannot be iteratively refined further. We choose PicoMUS as a comparison to our tool because it is a state-of-art MUS extractor, and the results it returned for our set of benchmarks are proved to be minimal. The data shows that in most of these cases, our tool can produce a minimal core. For the subset-3 benchmark, we obtain a different core of smaller size than the one obtained from PicoMUS. The results demonstrate the power of the Gröbner basis technique to identify the causes of unsatisfiability.

8 Conclusions

This paper addresses the problem of identifying an infeasible core of a set of multivariate polynomials, with coefficients from a field, that have no common zeros. The problem is posed in the context of computational algebraic geometry and solved using the Gröbner basis algorithm. We show that by recording the data produced by the Buchberger's algorithm – the $Spoly(f_i, f_j)$ pairs, as well as the polynomials of F used in the division process $Spoly(f_i, f_j) \xrightarrow{F}_+ 1$ – we can identify certain conditions under which a polynomial can be discarded from a core. An algorithm was implemented within the Singular computer algebra tool and experiments were conducted to validate the approach. While the use

of GB engines for SAT solving has a rich history, the problem of unsat core identification using GB-engines has not been addressed by the SAT community. We hope that this paper will kindle interest in this topic which is worthy of attention from the SAT community – particularly when there is a renewal of interest in the use of Gröbner bases for formal verification [21–24].

References

1. de Siqueira, J.L., Puget, J.-F.: Explanation-based generalization of failures. In: Proceedings of European Conference Artificial Intelligence, pp. 339–344 (1988)
2. Jiang, J.-H.R., Lee, C.-C., Mishchenko, A., Huang, C.-Y.: To SAT or Not to SAT: scalable exploration of functional dependency. IEEE Trans. Comp. **59**(4), 457–466 (2010)
3. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003)
4. Marques-Silva, J.: Minimal unsatisfiability: models, algorithms and applications. In: IEEE International Symposium on Multi-Valued Logic, pp. 9–14 (2010)
5. Nadel, A., Ryvchin, V., Strichman, O.: Accelerated deletion-based extraction of minimal unsatisfiable cores. J. Satisfiability Boolean Model. Comput. **9**, 27–51 (2014)
6. Belov, A., Lynce, I., Marques-Silva, J.: Towards efficient MUS extraction. AI Commun. **25**(2), 97–116 (2012)
7. Adams, W.W., Loustaunau, P.: An Introduction to Gröbner Bases. American Mathematical Society, Providence (1994)
8. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer, New York (2007)
9. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassen-ringes nach einem nulldimensionalen Polynomideal, Ph.D. Dissertation. Philosophiesche Fakultat an der Leopold-Franzens-Universitat, Austria (1965)
10. Clegg, M., Edmonds, J., Impagliazzo, R.: Using the Gröbner basis algorithm to find proofs of unsatisfiability. In: ACM Symposium on Theory of Computing, pp. 174–183 (1996)
11. Beame, P., Impagliazzo, R., Krajiček, J., Pitassi, T., Pudlák, P.: Lower bounds on Hilbert's Nullstellensatz and propositional proofs. Proc. Lond. Math. Soc. **73**, 1–26 (1996)
12. Condrat, C., Kalla, P.: A Gröbner basis approach to CNF-formulae preprocessing. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 618–631. Springer, Heidelberg (2007)
13. Zengler, C., Küchlin, W.: Extending clause learning of SAT solvers with Boolean Gröbner bases. In: Gerdt, V.P., Koepf, W., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2010. LNCS, vol. 6244, pp. 293–302. Springer, Heidelberg (2010)
14. Brickenstein, M., Dreyer, A.: Polybori: a framework for Gröbner basis computations with Boolean polynomials. J. Symbolic Comput. **44**(9), 1326–1345 (2009)
15. Vardi, M.Y., Tran, Q.: Groebner bases computation in Boolean rings for symbolic model checking. In: IASTED (2007)
16. van Loon, J.N.M.: Irreducibly inconsistent systems of linear inequalities. Eur. J. Oper. Res. **8**(3), 283–288 (1981)

17. Chinneck, J.W., Dravnieks, E.: Locating minimal infeasible constraint sets in linear programs. INFORMS J. Comput. **3**(2), 157–168 (1991)
18. Decker, W., Greuel, G.-M., Pfister, G., Schönemann, H.: Singular 3–1-3 — A computer algebra system for polynomial computations (2011). http://www.singular.uni-kl.de
19. Agnew, G.B., Mullin, R.C., Onyszchuk, I., Vanstone, S.A.: An implementation for a fast public-key cryptosystem. J. Cryptology **3**(2), 63–79 (1991)
20. Reyhani-Masoleh, A., Hasan, M.A.: Low complexity word-level sequential normal basis multipliers. IEEE Trans. Comput. **54**(2), 98–110 (2005)
21. Lv, J., Kalla, P., Enescu, F.: Efficient Gröbner basis reductions for formal verification of Galois field arithmetic circuits. IEEE Trans. CAD **32**(9), 1409–1420 (2013)
22. Gao, S., Platzer, A., Clarke, E.M.: Quantifier elimination over finite fields using Gröbner bases. In: Winkler, F. (ed.) CAI 2011. LNCS, vol. 6742, pp. 140–157. Springer, Heidelberg (2011)
23. Kalla, P.: Formal verification of arithmetic datapaths using algebraic geometry and symbolic computation. In: Invited Tutorial, Proceedings of FMCAD, p. 2 (2015). http://www.cs.utexas.edu/users/hunt/FMCAD/FMCAD15/slides/fmcad15-tutorial-kalla.pdf
24. Sayed-Ahmed, A., Große, D., Kühne, U., Soeken, M., Drechsler, R.: Formal verification of integer multipliers by combining Gröbner basis with logic reduction. In: Proceedings of Design Automation and Test in Europe, pp. 1048–1053 (2016)

The Power of Propagation: When GAC Is Enough

David A. Cohen[1] and Peter G. Jeavons[2(✉)]

[1] Department of Computer Science, Royal Holloway,
University of London, London, UK
`d.cohen@rhul.ac.uk`
[2] Department of Computer Science, University of Oxford, Oxford, UK
`peter.jeavons@cs.ox.ac.uk`

Considerable effort in constraint programming has focused on the development of efficient propagators for individual constraints: reducing the current domains of the variables of each constraint without removing any allowed assignments. The strongest consistency that such a reduction can establish is when each value in the current domains of every variable is part of an allowed assignment: assigning each other variable of the constraint a value from its current domain. When this condition holds the domains are said to satisfy the property of *generalised arc-consistency* (GAC) for that constraint.

The development of efficient GAC propagators for *individual* global constraints does not help determine the effectiveness of GAC propagation when applied to a network of *overlapping* global constraints. In this paper, we precisely characterise the classes of constraint problems where such propagators can decide the existence of a solution on their own, without the need for any additional search. We say that such classes are decided by GAC.

Sporadic examples of such classes have previously been identified, including classes based on restricting the structure of the problem, restricting the constraint types, and some hybrid examples. However, there has previously been no unifying theoretical framework which characterises all of these classes: structural, language-based and hybrid. In this paper we develop such a unifying approach based on the notion of tree-duality described in [1]. We then show how all of the known classes fit into this common framework.

One important application area for our results is to particular global constraints which can themselves be decomposed into combinations of smaller constraints. If the instance formed by these smaller constraints is decided by GAC, and retains this property when we add an arbitrary unary constant constraint, then we have an effective GAC propagator for the original global constraint.

Another application area is the identification of sub-problems of a given problem that can be solved efficiently by existing solvers, that can be used as targets for problem reduction or pre-processing strategies.

Abstract of Technical Track.

Reference

1. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. SIAM J. Comput. **28**, 57–104 (1998)

Constraint Programming for Planning Test Campaigns of Telecommunication Satellites

Emmanuel Hebrard[1]([✉]), Marie-José Huguet[1], Daniel Veysseire[1],
Ludivine Boche Sauvan[1,2], and Bertrand Cabon[2]

[1] LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
hebrard@laas.fr
[2] Airbus Defense and Space, Toulouse, France

Abstract. The payload of a satellite is the equipment that actually performs the mission, it receives, amplifies and emits back the signal. Before launch, the equipment of the payload must be tested under a simulated space environment. A set of tests are necessary for certifying that the payload can perform its mission in space. Each test is characterized by a set of activated equipment units and several thermal constraints limit the number of equipment units that can be activated simultaneously. The duration of the tests themselves is incompressible. However, activating some equipment takes time as each equipment unit much reach a given temperature and the temperature of the entire payload must be stabilized before tests can resume. Then, the transition time between tests depends on the order in which tests are sequenced.

Sequencing these tests in a way that ensures the thermal stability of the payload and minimizes the overall duration of the test campaign is a very important objective for satellite manufacturers.

To achieve these goals, we want first to minimize the number of times the configuration of the payload has to be changed at all, which can be modeled as a packing problem. Moreover, we also want to minimize the total number of times an equipment unit has to be activated. We model this second aspect using the constraint SWITCH, where the set of active equipment is viewed as a buffer with a capacity and switches of buffered items are to be minimized.

We introduce implied constraints that improve the lower bound of the model for the number of configurations, and an efficient search strategy for this problem. Moreover, we propose an improvement of the propagation algorithm for SWITCH which takes into account items that must necessarily be visited, however at possibly unknown positions in the sequence.

Finally, we experimentally evaluate the different contributions and assess the benefit of our method with respect to the approach currently implemented at Airbus.

Abstract of Application Track.

Graphical Models for Optimal Power Flow

Krishnamurthy Dvijotham[1]([✉]), Pascal Van Hentenryck[2],
Michael Cherkov[1,2,3], Sidhant Misra[1,2,3], and Marc Vuffray[3]

[1] Computing and Mathematical Sciences,
California Institute of Technology, Pasadena, USA
dvij@cs.washington.edu
[2] Industrial and Operations Engineering,
University of Michigan, Ann Arbor, USA
[3] T-division, Los Alamos National Lab, Los Alamos, USA

As the penetration of renewable energy sources in the power grid increases, the availability of generation capacity becomes increasingly stochastic (as it depends on unpredictable weather conditions like wind and sunlight). Operating a reliable power grid in this scenario will require smart management of demand-side resources (in order to achieve balance between generation and demand in real time), leading to the need for efficient optimization algorithms that can quickly compute the best utilization of distributed flexible demand-side resources. The core underlying mathematical problem is the Optimal power flow (OPF) problem: Minimize the cost of resources used subject to network constraints. Although solved routinely in the course of power grid operations, it is known to be strongly NP-hard in general, and weakly NP-hard on tree-structured networks. Further, it is typically solved in a centralized manner that is not suitable for distributed energy resources whose availability and cost is unpredictable.

In this paper, we take initial steps towards developing an efficient distributed optimization framework for OPF. We focus on tree networks, because power distribution networks which connect power consumers to the high-voltage transmission network are typically tree-structured. We formulate the OPF problem over tree networks (power distribution networks linking consumers to the main transmission grid as an inference problem over a tree-structured graphical model (GM) where the nodal variables are low-dimensional vectors. We adapt the dynamic programming (DP) algorithm for inference over a tree-structured GM to the OPF problem. The resulting DP formulation is intractable, due to the need to pass infinite-dimensional messages (functions of the nodal variables). We remedy this problem by performing an interval discretization of the nodal variables and use CP techniques to adaptively tighten bounds on variables as the DP algorithm proceeds up the tree. The final result is an algorithm that computes an ϵ-feasible (all constraints are satisfied upto tolerance ϵ) super-optimal (objective value better than the true optimum) solution to the OPF problem in time linear in the network size and polynomial in $\frac{1}{\epsilon}$. The algorithm can be implemented in a distributed manner that only requires communication between

---

Abstract of Computational Sustainability Track.

neighboring nodes in the network. Compared to previous algorithms that solve OPF with optimality guarantees using convex relaxations, our approach is able to work for arbitrary distribution networks and handle mixed-integer optimization problems. We evaluate our technique numerically on several benchmark networks and show that practical OPF problems can be solved efficiently using this approach. We outline other applications of the framework (probabilistic analysis of power grids, for example) and ideas to extend the approach to meshed networks.

"Almost-Stable" Matchings in the Hospitals/Residents Problem with Couples

David F. Manlove[(⊠)], Iain McBride, and James Trimble

School of Computing Science, Sir Alwyn Williams Building,
University of Glasgow, Glasgow G12 8QQ, UK
david.manlove@glasgow.ac.uk

Abstract. The Hospitals/Residents problem (HR) models the problem of allocating junior doctors (or *residents*) to hospitals, where residents have preferences over the hospitals they would like to work in, and vice versa, hospitals have preferences over the residents that they would employ. The Hospitals/Residents problem with Couples (HRC) is an extension of HR in which couples (comprising pairs of residents) are allowed to submit joint preference lists over pairs of (typically geographically close) hospitals. In this context we seek an allocation of residents to hospitals that is *stable*, which guarantees that no resident and hospital, and no couple and pair of hospitals, have an incentive to deviate from their assignments and become assigned to each other outside of the matching.

It is known that a stable matching need not exist in an instance of HRC, and moreover, the problem of determining whether an instance of HRC admits a stable matching is NP-complete. To cope with the possible non-existence of a stable matching, we consider the problem of finding a matching that is "as stable as possible", i.e., admits the minimum number of blocking pairs. We denote this problem by MIN BP HRC.

We show that MIN BP HRC is NP-hard and very difficult to approximate, even in the highly restricted case that each couple finds only one hospital pair acceptable. However if we further assume that the preference list of each single resident and hospital is of length at most 2, we give a polynomial-time algorithm for MIN BP HRC. We then present the first Integer Programming (IP) and Constraint Programming (CP) models for the general case of MIN BP HRC. Finally, we discuss an empirical evaluation of these models applied to randomly-generated instances of MIN BP HRC. We find that on average, the CP model is about 1.15 times faster than the IP model, and when presolving is applied to the CP model, it is on average 8.14 times faster. We further observe that the number of blocking pairs admitted by a solution is very small, i.e., usually at most 1, and never more than 2, for the (28,000) instances considered.

The full version of this paper has been submitted for fast track publication in *Constraints*.

Mixed-Integer and Constraint Programming Techniques for Mobile Robot Task Planning

Kyle E.C. Booth[✉], Tony T. Tran, Goldie Nejat, and J. Christopher Beck

Department of Mechanical and Industrial Engineering,
University of Toronto, Toronto, ON M5S 3G8, Canada
{kbooth,tran,nejat,jcb}@mie.utoronto.ca

In this work we investigate the application of optimization-based scheduling technologies to two mobile robot task planning problems. We develop and apply *mixed-integer programming* (MIP) and *constraint programming* (CP) techniques to model and solve these problems, demonstrating through simulation that our methods outperform those previously proposed in the robotics literature. Our simulation further indicates that the inference-based search of CP is the superior approach for the problems studied. Additionally, we implement our CP approach for the second problem as a task planning module within a real robotics platform on the social robot Tangy, validating the physical utility of the method.

In the first task planning problem, a robot plans a set of tasks each with temporal constraints identifying when a task is available for execution and when it must be completed. The task planning module must determine a feasible plan while minimizing the sum of task completion times. Existing approaches use heuristic decisions aimed at reducing problem complexity and improving runtime performance at the cost of optimality. Our methods provide better solutions in shorter runtimes without sacrificing completeness. The second task planning problem involves a socially-assistive robot facilitating everyday activities for retirement home residents. The robot must create task plans while reasoning about temporal constraints, human user timetables, and robot energy levels. We show that our optimization-based methods outperform a previously proposed forward-chaining temporal planning approach, and integrate our CP technique into a task planning module within a mobile robotics platform architecture.

Overall, our results indicate that these optimization-based techniques are promising for solving mobile robot task planning problems. A primary direction for our future research is to investigate the role of these methods for the development of re-planning and plan repair techniques, in efforts to enhance the capability of our task planning module. We plan to further investigate robot task planning problems in order to understand the point at which such problems will require more sophisticated strategies, such as customized search manipulations and problem decompositions.

The Power of Arc Consistency for CSPs Defined by Partially-Ordered Forbidden Patterns

Martin C. Cooper[1] and Stanislav Živný[2][(✉)]

[1] IRIT, University of Toulouse III, 31062 Toulouse, France
cooper@irit.fr
[2] Department of Computer Science, University of Oxford, Oxford, UK
standa.zivny@cs.ox.ac.uk

Characterising tractable fragments of the constraint satisfaction problem (CSP) is an important theoretical challenge. Forbidding patterns (generic sub-instances) provides a means of defining CSP fragments which are neither exclusively language-based nor exclusively structure-based. It is known that the class of binary CSP instances in which the broken-triangle pattern (BTP) does not occur, a class which includes all tree-structured instances, are decided by arc consistency (AC), a ubiquitous reduction operation in constraint solvers. We provide a characterisation of simple partially-ordered forbidden patterns which have this AC-solvability property. It turns out that BTP is just one of five such patterns The four other patterns allow us to exhibit new tractable classes, including one which strictly generalises binary max-closed CSPs (the pattern EMC).
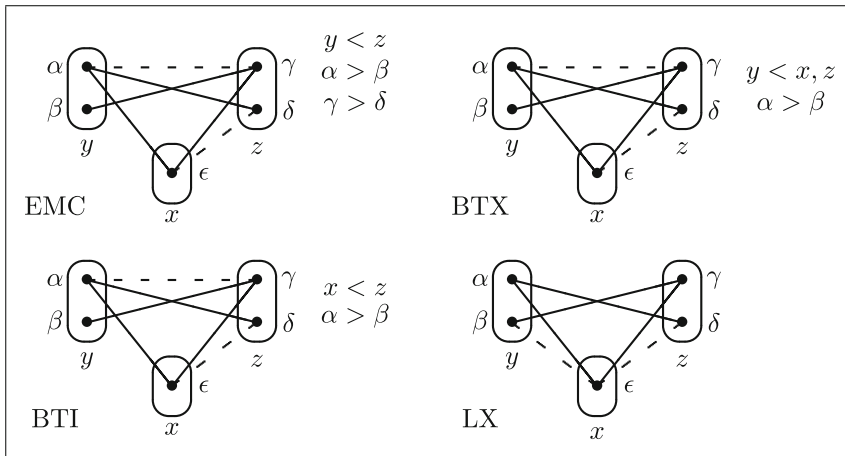
An *ordered pattern* is a binary CSP instance in which the consistency of some assignments to pairs of variables is undefined, with an order on the domains and

the variables. A pattern P *occurs* in an instance (or pattern) I if there is a homomorphism from P to I via an injective renaming of variables. $\mathrm{CSP}_{\overline{SP}}(P)$ denotes the set of binary CSP instances in which the pattern P does not occur.

Theorem 1. *If P is a simple pattern with a partial order on its domains and/or variables, then $CSP_{\overline{SP}}(P)$ is AC-solvable if and only if P occurs in one of the patterns EMC, BTX, BTI, LX or BTP (or patterns which are equivalent).*

DASH: Dynamic Approach for Switching Heuristics

Giovanni Di Liberto[1], Serdar Kadioglu[2(✉)], Kevin Leo[3], and Yuri Malitsky[1]

[1] Cork Constraint Computation Centre, University College Cork, Cork, Ireland
dilibert@dei.unipd.it, yuri.malitsky@gmail.com
[2] Oracle Corporation, Burlington, MA 01803, USA
serdar.kadioglu@oracle.com
[3] Faculty of IT, Monash University, Melbourne, Australia
kevin.leo@monash.edu

Complete tree search is a highly effective method for tackling combinatorial optimization and satisfaction problems. Over the years, a plethora of branching heuristics have been introduced to further refine the technique for varying problems. Yet while each new approach continued to push the state-of-the-art, parallel research began to repeatedly demonstrate that there is no single method that would perform the best on all problem instances. Tackling this issue, portfolio algorithms took the process a step further, by trying to predict the best heuristic for each instance at hand. However, the motivation behind algorithm selection can be taken further still, and used to dynamically choose the most appropriate algorithm for each encountered sub-problem.

In this paper [1], we introduce a method that advances the idea of instance-specific algorithm configuration to the fine-grained level of sub-instance configuration and generalized dynamic heuristic selection. We identify a feature space that captures both the evolution of the problem in the branching tree for Mixed-Integer Programming (MIP) problems and the similarity among sub-problems of instances from the same model. We monitor the tree-search and present empirical evidence that MIP instances gradually change structure as branching heuristics are applied. Tracing the sub-problems during search allows dynamic heuristic selection and we show how to exploit this information on-the-fly in order to decide the best time to switch the branching variable selection heuristic.

Experiments on a highly heterogeneous collection of hard MIP instances show significant gains over the standard pure approach which commits to a single heuristic throughout the search. We conclude that dynamic heuristic selection can be very beneficial, yet the existence of complementary heuristics, diverse set of instances, and descriptive features are also important. Since our method is generally applicable to complete tree search, it can be readily extended to closely related paradigms such as Constraint Programming and Boolean Satisfiability.

Reference

1. Di Liberto, G., Kadioglu, S., Leo, K., Malitsky, Y.: Dash: Dynamic approach for switching heuristics. Eur. J. Oper. Res. **248**(3), 943–953 (2016)

Abstract of Journal-First and Sister Conferences Track.

AHP Based Portfolio Selection with Risk Preference Modeling

Cristinca Fulga[(✉)]

Bucharest University of Economic Studies,
Piata Romana 6, 010374 Bucharest, Romania
`fulga@csie.ase.ro`

Abstract. We propose a methodology for defining, measuring and optimizing risk that addresses some of the conceptual shortcomings of the *Mean-Risk* framework for the portfolio problem such as the disregard of the investor's attitude towards risk and implicit assumption of neutrality to loss aversion, see Fulga (2016). Firstly, we define a risk measure that, for continuous return distribution functions, can be represented in terms of the conditional expectation of the distribution tail, where the tail is determined by the critical return level θ characterizing the loss-averse investor. We give equivalent forms of the proposed risk measure and point out the relations with $CVaR$, and *LPM of first order* (LPM_1), establish its properties and study the link with stochastic dominance criteria. We discuss practical aspects regarding the calculation in the case of scenario-based portfolio optimization. Secondly, we propose a new methodology for portfolio selection in which investor's loss aversion is fully taken into consideration. Three types of investors characterized by different classes of utility functions with loss aversion are considered. Thirdly, we perform an empirical study which is targeted on assessing the differences between the efficient frontier of the proposed model and the classical $Mean\text{-}Variance$, $Mean\text{-}CVaR_\alpha$ and $Mean\ \text{-}LPM_1$ frontiers. We analyze the loss of welfare incurred by using another model instead of the proposed one and measure the gain/loss of utility incurred by choosing it. We use a proximity index of the *welfare gain/loss*. Then, we assess how much the portfolios really differ in terms of their compositions and use a dissimilarity index based on the 1-norm. Two methods are used to evaluate the proximity of two efficient frontiers. We describe and interpret the optimal solutions obtained with the proposed model and emphasize the role and influence of loss aversion parameters values and of constraints.

Reference

1. Fulga, C.: Portfolio optimization under loss aversion. Eur. J. Oper. Res. **251**(1), 310–322 (2016). doi:10.1016/j.ejor.2015.11.038

Abstract of Journal-First and Sister Conferences Track.

STR3: A Path-Optimal Filtering Algorithm for Table Constraints

Christophe Lecoutre[1,2]([✉]), Chavalit Likitvivatanavong[1,2], and Roland H.C. Yap[1,2]

[1] CRIL-CNRS UMR 8188, Université d'Artois, 62307 Lens, France
[2] School of Computing, National University of Singapore, Central Area, Singapore
lecoutre@cril.fr, likitchav@gmail.com, ryap@comp.nus.edu.sg

Constraint propagation is fundamental to Constraint Programming (CP). Many algorithms have been proposed over the years to enforce the property called Generalized Arc Consistency (GAC) — the most used form of propagation on many types of constraints, including table constraints. Out of the many GAC algorithmic ideas for table constraints, an approach called *simple tabular reduction* (STR) which maintains the tables of constraints by removing invalid tuples has been show to be effective. In particular, STR2, a refined STR variant is among the most efficient GAC algorithms for positive table constraints.

In this paper, we revisit this approach by proposing a new GAC algorithm called STR3 that is specifically designed to enforce GAC during backtrack search. Thus, STR3 is a *maintaining (generalized) arc consistency* (MAC) algorithm, making it different from most other propagation algorithms, including STR2. STR3 employs different data structures from the STR2 (and STR1) algorithm which allows a tuple to be found with respect to a domain value without visiting irrelevant tuples.

Most of the GAC algorithms for table constraints previously introduced in the literature suffer from having to repeatedly traverse the same tables or related data structures during search. STR3 is designed specifically to be interleaved with backtracking search, it's main goal is to maintain the consistency while minimizing the cost of backtracking. An important property of STR3 is that it can completely avoid unnecessary traversal of tables, making it optimal along any path of the search tree, i.e. *path-optimal*. We also study a variant of STR3, based on an optimal circular way for traversing tables, and discuss the relationship of STR3 with two other optimal GAC algorithms introduced in the literature, namely, GAC4 and AC5TC-Tr.

Finally, we demonstrate experimentally how STR3 is competitive with the state-of-the-art. In particular, our experiments show that the effectiveness of STR3 is complementary to STR2. STR2 is faster than STR3 where simple tabular reduction can eliminate a large number of tuples from the tables that they become small. STR3, by contrast, outperforms STR2 when constraint relations do not shrink as much during search.

Boosting Symmetry Breaking During Search in Constraint Programming

Jimmy H.M. Lee[(✉)] and Zichen Zhu

Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{jlee,zzhu}@cse.cuhk.edu.hk

1   Extended Abstract

Symmetries can be broken statically or dynamically. Advantages of dynamic symmetry breaking methods include ability to break symmetries of arbitrary kinds and compatibility with variable and value heuristics. When considering runtime, dynamic methods often lose out to widely used static methods, such as the LexLeader method and value precedence. The focus of this paper is dynamic symmetry breaking, in particular the symmetry breaking during search (SBDS) method that adds conditional symmetry breaking constraints during search.

We first propose *Recursive SBDS (ReSBDS)* [1] to improve the pruning power of partial SBDS. The main idea is to add extra symmetry breaking constraints during search recursively to prune also symmetric nodes of some pruned subtrees. Our proposal features a careful tradeoff between the number of constraints added and the benefits of extra pruning. We give theoretical characterization on the soundness and termination of our method, and comparisons on pruning strengths against other well-known symmetry breaking methods, such as LDSB and the LexLeader method.

Symmetry breaking constraints added by SBDS and its variants are actually nogoods. To reduce the overhead of nogood propagation, we propose weak-nogood consistency (WNC) [2], a weaker consistency notion than generalized arc consistency (GAC) for nogoods to trade pruning power for efficiency. We present an efficient lazy propagator to enforce WNC for SBDS (and its variants) using *one* watched literal. A similar weaker consistency, generalized weak-incNGs consistency (GWIC), together with a lazy propagator is also proposed for the incNGs constraint. By exploiting the increasing property of the nogoods in inc-NGs, our lazy propagator watches also one literal for each global constraint, and operates and benefits from a similar lazy principle. We give formal characterization of the pruning strengths of the proposed propagators, and also the space and time complexities.

Abstract of Journal-First and Sister Conferences Track.

References

1. Lee, J., Zhu, Z.: Boosting SBDS for partial symmetry breaking in constraint programming. In: AAAI 2014, pp. 2695–2702 (2014)
2. Lee, J., Zhu, Z.: Filtering nogoods lazily in dynamic symmetry breaking during search. In: IJCAI 2015, pp. 339–345 (2015)

Enhancing Partial Symmetry Breaking in Constraint Programming

Jimmy H.M. Lee[✉] and Zichen Zhu

Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{jlee,zzhu}@cse.cuhk.edu.hk

1 Extended Abstract

Symmetries are common in many constraint problems. They can be broken statically or dynamically. Static methods alter the original problem by adding new constraints to remove symmetric solutions. In contrast, dynamic methods modify the search procedure to exclude exploration of symmetric regions. Practical symmetry breaking methods often trade completeness for efficiency by only breaking a subset of symmetries. The pruning power of partial symmetry breaking depends on the given subset of symmetries to break as well as the extra composition symmetries that are broken by the interactions among symmetry breaking constraints.

In the context of Partial Symmetry Breaking During Search (ParSBDS), the search order affects the extra composition symmetries. In this paper [1], we give the first formal characterization of the pruning behavior of ParSBDS and its state-of-the-art variants. Relying on a generalization of (symmetry) dominance, we prove that ParSBDS and variants prune all and only nodes that are dominated. We introduce the notion of Dominance-Completeness (DC-ness), show that ParSBDS and variants eliminate the symmetry group of the given subset of symmetries if the resultant search tree is DC, and give a situation where this happens. Unfortunately, building a DC tree is not always possible. We approximate it using two search heuristics.

LexLeader, the state of the art static method, adds lex ordering constraints to break symmetries. We propose a new total ordering, reflex [2], and also a reflex ordering constraint for breaking variable symmetries. An efficient GAC filtering algorithm is given for reflex ordering. Based on this reflex ordering, the ReflexLeader method is thus proposed by adding one reflex ordering constraint for every variable symmetry of the problem or a subset. We show empirically that given the same subset of symmetries, ReflexLeader can break more composition symmetries than LexLeader. We prove that ReflexLeader is safe to combine with Precedence and multiset ordering constraints.

References

1. Lee, J.H.M., Zhu, Z.: Breaking more composition symmetries using search heuristics. In: AAAI 2016, pp. 3418–3425 (2016)
2. Lee, J.H.M., Zhu, Z.: Static symmetry breaking with the reflex ordering. In: IJCAI 2016 (2016, to appear)

Decomposition of the Factor Encoding for CSPs

Chavalit Likitvivatanavong[✉], Wei Xia, and Roland H.C. Yap

School of Computing, National University of Singapore, Singapore, Singapore
likitchav@gmail.com, {xiawei,ryap}@comp.nus.edu.sg

Generalized arc consistency (GAC) is one of the ways for reducing the search space when solving constraint satisfaction problems (CSPs). With so many GAC algorithms having been developed, GAC is invariably implemented in most solvers in one form or another. Consistencies stronger than GAC can further reduce the search space and have also been shown useful, but designing and implementing such algorithms to be competitive with the state-of the-art solvers employing GAC turns out to be a challenge. Higher-order consistency properties include relational consistency, max-restricted pairwise consistency, and full pairwise consistency (FPWC).

An alternative approach to avoid the need for an efficient implementation is to convert a CSP into another CSP and apply existing GAC propagators on the result, such that it is equivalent to enforcing the stronger consistencies on the original CSP. Several such CSP transformations have been proposed. Among them, the factor encoding (FE) [2], which extracts commonly shared variables between pairs of constraints forming new variables called *factor variables*, achieves FPWC on the original CSP. However, although FE is an efficient way of achieving FPWC, many new factor variables can be added which can substantially increase the size of the CSP along with increased overheads.

In [1], we address the shortcoming of the FE with the *factor-decomposition encoding* (FDE). FDE decomposes constraints such that factor variables and their corresponding original variables are taken out to form new constraints. We show that the FDE preserves the main property of the FE. In many instances, the FDE also decreases the arity of the CSP compared with FE. We perform an experimental study on FE and FDE using multiple search heuristics. Our experiments show that the change in constraint network from FDE can benefit dynamic search heuristics. The FDE is competitive with the FE on a majority of problem instances. It can reduce the search space and speed up the solving on some structured problems by orders of magnitude compared with GAC.

References

1. Likitvivatanavong, C., Xia, W., Yap, R.H.C.: Decomposition of the factor encoding for CSPs. In: IJCAI, pp. 353–359 (2015)
2. Likitvivatanavong, C., Xia, W., Yap, R.H.C.: Higher-order consistencies through GAC on factor variables. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 497–513. Springer, Heidelberg (2014)

This is a summary of paper [1].

Abstract of Journal-First and Sister Conferences Track.

Tightness of LP Relaxations for Almost Balanced Models

Adrian Weller[1]([⊠]), Mark Rowland[1], and David Sontag[2]

[1] University of Cambridge, Cambridge, UK
{aw665,mr504}@cam.ac.uk
[2] New York University, New York, NY, USA
dsontag@nyu.edu

We examine Boolean binary weighted constraint satisfaction problems and derive sufficient conditions for when certain linear programming (LP) relaxations are guaranteed to return an integer solution, in which case the solution is exact and we say that the relaxation is *tight*.

Specifically, we are interested in the problem of finding a configuration of variables $x = (x_1, \ldots, x_n) \in \{0, 1\}^n$ that maximizes a score function, defined by unary and pairwise rational terms $f(x) = \sum_{i=1}^n \psi_i(x_i) + \sum_{(i,j) \in E} \psi_{ij}(x_i, x_j)$. In the machine learning community, this is typically known as *MAP* (or *MPE*) *inference*, yielding a configuration of variables with maximum probability.

We derive sufficient *hybrid conditions* (combining restrictions on both structure and on the allowed language of cost functions [1]) for guaranteed tightness of (i) the basic LP relaxation on the local polytope LP+LOC, and (ii) the LP relaxation on the triplet-consistent polytope LP+TRI (the next level in the Sherali-Adams hierarchy). We provide simple new proofs of earlier results and derive significant novel results including that LP+TRI is tight for any model where each block is *balanced* or *almost balanced*, and a decomposition theorem that may be used to break apart complex models into smaller pieces.

A balanced (sub-)model contains no frustrated cycles. An almost balanced (sub-)model contains no frustrated cycles except through one privilegedbreak variable.

LP+TRI was known to be tight for any model with treewidth 2. Combining results shows that LP+TRI dominates a recent method involving a reduction to the maximum weight stable set problem on a derived graph related to the microstructure complement of the dual representation (featured in the published journal paper track of CP 2015 [2, 3, 4]), in the sense that LP+TRI is guaranteed to solve a strict superset of problems for any valid score functions in polynomial time.

References

1. Cooper, M., Živný, S.: Hybrid tractability of valued constraint problems. Artif. Intell. **175**(9), 1555–1569 (2011)
2. Jégou, P.: Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In: AAAI, pp. 731–736 (1993)
3. Larrosa, J., Dechter, R.: On the dual representation of non-binary semiring-based CSPs. In: CP 2000 Workshop on Soft Constraints (2000)
4. Weller, A.: Revisiting the limits of MAP inference by MWSS on perfect graphs. In: Artificial Intelligence and Statistics (AISTATS) (2015)

Author Index