Speaker: Zion Schell *Monday, April 15, 2013*
Scribe: Fikayo Adetunji
Topic: Tractable Constraint Languages by David Cohen and Peter Jeavons
Reference: Chapter 11 of the CP book by Rina Dechter

Introduction

The speaker Zion started the class by alerting us that the topic was significantly different from anything we had seen before in CP. For that reason, his goal was to acquaint us with the contents of the chapter rather than explaining it to us. Having read the chapter several times, I agreed and thought it was the best that anyone could do.
He started his presentation by motivating the topic and providing some definitions. After which, he introduced the *expressiveness* and *complexity* of constraint languages. Finally, he presented the idea of hybrid tractability.

Motivation

- *Constraint solvers* allow a user to define and solve constraint networks by providing a set of basic constraints, or constraint primitives, to be declared over variables.
- The set of constraint primitives form the *constraint language* of the solver and restrict the set of constraint problems handled by the solver.
- Increasing the *expressiveness* of the language may increase the *complexity* of a solver, and that of the algorithms embedded in the solver.
- As a result the design of a constraint language requires a balance between performance and expressiveness.
- This chapter discusses the *expressiveness* of *tractable* constraint languages.

Basic Definitions

- A *constraint language* is a set of relations.
- The *relational subclass* of a constraint language is the set of all CSP instances that use only relations from the language.
- *Tractability* can be expressed in two ways:
  - A tractable constraint language that has a polynomial algorithm for solving all instances using the relational subclass.
  - A relation R is tractable if the language formed by {R} is tractable.
- The authors relate tractability to constraint arity and domain size of a CSP as shown in Table 1.

Table 1: A table showing the tractability of some problems.

Problem type	Domain size	Constraint arity	Tractability
2SAT	2	2	Tractable
3SAT	3	2	Intractable
Graph 3-coloring	2	3	Intractable

- The constraint arity, domain size, constraint types of a constraint language determine its tractability. Figure 1 illustrates that a problem with a domain size less than or equal to two is tractable, else its tractability is not

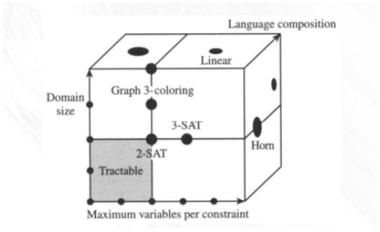


Fig. 1: A cube that depicts tractability.

guaranteed. For example, Horn-SAT is tractable although it allows constraints of arbitrary arity.

CHiP: An example of an existing constraint language

- It is used for constraint handling in Prolog.
- Variables' domains are subset of $\mathbb{N}$ (i.e., natural numbers), and are theoretically infinite.
- Constraints can be of the following types:
  - Domain constraints (unary): E.g., $x \geq a$; $x \leq a$
  - Arithmetic constraints (unary or binary): E.g., $ax \neq b$; $ax = by + c$; $ax \leq by + c$; $ax \geq by + c$.
  - Compound-arithmetic constraints (*n*-ary with arbitrary arity): E.g., $a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq by + c$, $ax_1x_2\ldots x_n \geq by + c$, $(a_1x_1 \geq b_1) \vee (a_2x_2 \geq b_2) \vee \ldots \vee (a_nx_n \geq b_n) \vee (ay \leq b)$
- Enforcing arc-consistency in CHiP allows the generation of backtrack-free solutions because CHiP is tractable.
- Thus, although in CHIP the constraint arity and domain size are not restricted, the language remains tractable. Thus, there is more to tractability than the restriction of domain size and/or constraint arity to two.

More examples of a tractable language: The constant language

- The constant language
  - It has a domain set with a single element: 0.
  - It consists of relations that either equal or do not equal 0: $\{(x=0),(x=y=0),(x=y=z=0),\ldots\}$, $\{(x\neq 0)\}$.
  - Constraints are tested by checking whether variables equal zero. If any variable fails, then there is no solution.
- The max-closed language
  - The domain is a linearly ordered set. Given x and y in the set, one of x and y is necesarily greater than the other, i.e., there is no tie between x and y.
  - The language requires any max-closed relations on the domain that are based on the function max(a,b)
    - Can be extended to tuples element-wise: $\max((a_1,a_2),(b_1,b_2)) = (\max(a_1,b_1),\max(a_2,b_2))$.
    - Can be expressed with the function's domain closed. For example: (1,2) and (3,4) in the domain implies $(2,4) = \max((1,2),(3,4))$ in the domain.
- Horn-SAT
  - The variables are Boolean and the domains are $\{0,1\}$.
  - Each constraint is a disjunction of negated literals with at most one positive literal.
  - A Horn-SAT theory is solvable in P by unit resolution. It can be solved by transforming all disjunctions into implications (with only positive literals in each implication), where every rule can be fired at most once.

Expressiveness of constraint languages

- *Gadgets* are used in the construction of an expressible relation.

- Gadgets are CSPs (can be defined with variables, domains, and constraints) that extend a language beyond of what it strictly contains.
- Making new gadgets out of other gadgets is a task that must be achieved logically.
- An example of a gadget is the gadget for equality. This problem is depicted in Figure 2, and has a domain of three values {r,g,b}.
    - The task is to determine the relation between A and B.
    - The relation is A=B. The original language does not have equality, but the use of this gadget allows us to extend the language to express equality.
    - (A,B) is called the *construction site* of the problem.
- A gadget is used to define relation, while a *construction site* is the list of variables that the relation is projected on.

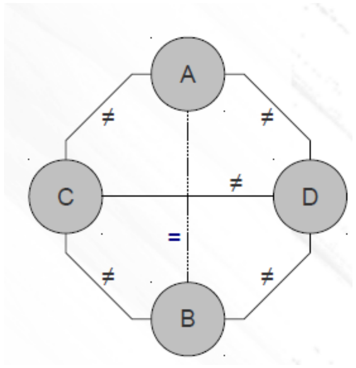


Fig. 2: Gadget of equality.

Expressiveness of constraint languages: k^th-order universal gadget of a constraint language Q ($U_k(Q)$)

- The domain of $U_k(Q)$ has the same domain as all problems in the relational subclass of Q. For example: If Q = {(A=0),(A=1),(A=2)}, then domain($U_k(Q)$) = {0,1,2}.
- The variables of $U_k(Q)$ has one variable for each k-tuple composed of elements in the domain of $U_k(Q)$. For example: If k=2 and domain($U_k(Q)$) = {0,1,2}, then the variables of $U_k(Q)$ = {v00,v01,v02,v10,v11,v12,v20,v21,v22}.
    - The name of a variable is the tuple to which it corresponds. For example: the name of v01 is (0,1).
    - The name relation of a list of variables is defined element-wise by variable names. For example: (v02,v01,v10,v22) → {(0,0,1,2),(2,1,0,2)}. Figure 3 better explains how the name relation is gotten from the variable names.

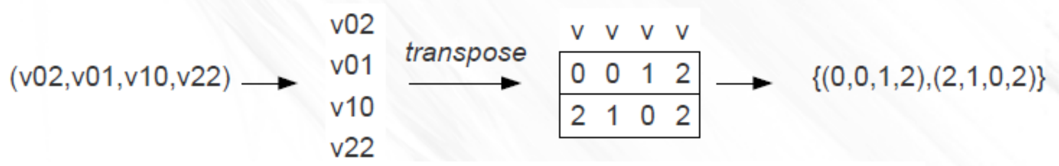
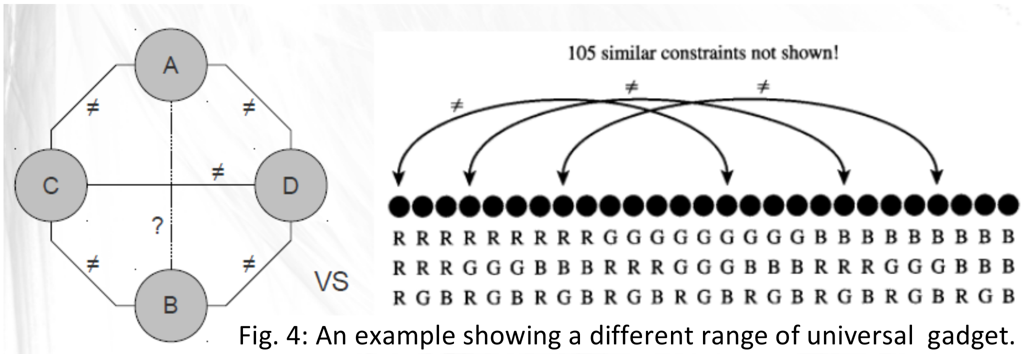


Fig. 3: How to obtain the name relation from the variable names.

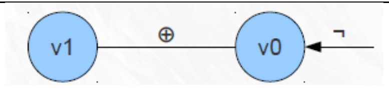
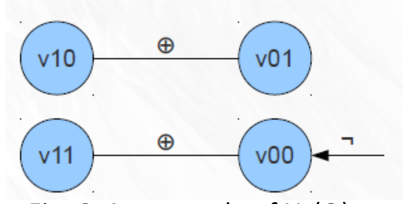
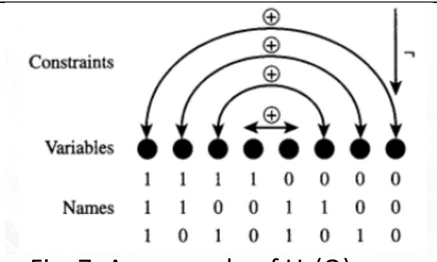
- The relations of $U_k(Q)$: for each relation R in Q, there is a need to find all tuples of variables so that when it is written vertically, the rows spell out some of the tuples of R. R is applied to a tuple of variables in $U_k(Q)$, if and only if the name relation of the tuple of variables is a subset of R.
- The universal gadget is not the smallest gadget for all applications. An example can be seen in Figure 4.


A
B
C
D
≠
?
VS
105 similar constraints not shown!
R R R R R R R R R G G G G G G G G G B B B B B B B B B
R R R G G G B B B R R R G G G B B B R R R G G G B B B
R G B R G B R G B R G B R G B R G B R G B R G B R G B
Fig. 4: An example showing a different range of universal gadget.

- The size of the universal gadget $U_k(Q)$ with domain size d is d^k.
- $U_k(Q)$ can also be used to determine the tractability of Q.
- The possibility of finding better gadgets in a given constraint language can be further more researched. If the language can be reduced, solutions will be found faster, since better gadgets results in faster systems.

Expressiveness of constraint languages: Examples involving $U_k(Q)$

Given Q = {A⊕B,¬A}: where A⊕B is the "xor" relation ((A,B) in {(0,1),(1,0)} satisfies the constraint); ¬A is the "not" relation ((A) in {(0)} satisfies the constraint); and the domain of $U_k(Q)$ is Boolean.

$U_k(Q)$	Variables (domain elements)	⊕ constraint	¬ constraint	Figured examples
$U_1(Q)$	1-tuples	matches (v0,v1) and (v1,v0), i.e., the name relation of (v0,v1) is {(0,1)}	matches (v0), i.e., the name relation of (v0) is {(0)}	 Fig. 5: An example of $U_1(Q)$.
$U_2(Q)$	2-tuples	matches (v00,v11), (v01,v10), (v10,v01), (v11,v00) i.e., the name relation of (v00,v11) is {(0,1),(0,1)}	matches (v00), i.e., the name relation of (v00) is {(0),(0)}	 Fig. 6: An example of $U_2(Q)$.
$U_3(Q)$	3-tuples	matches (v000,v111), (v001,v110), (v010,v101), (v011,v100), (v100,v011), (v101,v010), (v110,v001), (v111,v000) i.e., the name relation of (v001,v110) is {(0,1),(0,1),(1,0)}.	matches (v000), i.e., the name relation of (v000) is {(0),(0),(0)}	 Constraints ⊕ ⊕ ⊕ ⊕ ¬ Variables Names 1 1 1 1 0 0 0 0 / 1 1 0 0 1 1 0 0 / 1 0 1 0 1 0 1 0 Fig. 7: An example of $U_3(Q)$.

Expressiveness of constraint languages: Theorem 11.1 (Cohen, Gyssens, Jeavons 1996)

Let Q be a constraint language over a finite domain D, and let R be any relation over D.

Let k be the number of supports in R, and let L_R be any list of variables in $U_k(Q)$ whose name relation is R.

- Either $U_k(Q)$ expresses R as a gadget with construction site L_R.
- Or R is not expressible in Q.
- An example is the determination of how to express (A⇒B) with the constraint language Q = {A⊕B,¬A}
  - (A⇒B) formally: (A,B) in {(0,0),(0,1),(1,1)} satisfies the constraint.
  - (A⇒B) has 3 supports, so we use $U_3(Q)$.
- With the use of $U_3(Q)$
  - Look for a pair of variables with the name relation of which is a subset of {(0,0),(0,1),(1,1)} from Figure 7. Either the subset expresses the relations as a gadget or it can't be done.
  - Possible answers could be ((0,0,1),(0,1,1)), ((1,0,0),(1,0,1)), i.e., the the 7th and 5th variables or the 4th and 3rd variable.
  - If $U_3(Q)$ is treated as a gadget with the construction site (v100,v101), the resulting viable pairs of relation formed are: {(0,0),(0,1),(1,0),(1,1)}. We get four viable pairs because, if one of the variables is set as 1, the other has to be 0 or 1
  - The gadget does not form the relation (A⇒B).
  - It is thus proven that it is not possible to form (A⇒B) from{A⊕B,¬A} i.e., any pair of variables chosen would have always failed. There is no need to review all variables, only one is enough.
- The proof of this theorem is very complicated and could be studied from the textbook (pages 310-311).

Complexity of constraint languages: Theorem 11.2 (Schaefer 1978)

Let Q be a Boolean constraint language. Q is tractable if and only if for each R in Q:

- R allows (0,0,…,0) or (1,1,…,1)
- R allows disjunctive clauses with at most one negated variable (anti-horn clauses)
- R allows disjunctive clauses with at most one non-negated variable (horn clauses)
- R allows disjunctive clauses with at most two variables per clause (2-SAT)
- R is a set of solutions to linear equations on {0,1}

While the knowledge of a constraint language with a domain size of two is easy, it is difficult for other domain sizes.

Complexity of constraint languages: Necessary condition for tractability over a finite domain

A necessary condition for a tractable constraint language depends on the classifications of the possible types of solutions to the indicator problems, which, are most easily expressed using the notion of k-ary operation:

- A *k-ary operation* from D^k to D maps all k-tuples of elements of D to members of D. For example:
  - addition function (k=2, D=$\mathbb{N}$): +(a,b) maps (a,b) to a+b
  - maximum function (k=2, D=$\mathbb{R}$): max(a,b) maps (a,b) to a or to b
- *Idempotent* k-ary operation maps $(x,x,\ldots,x)$ to x for all x. For example:
  - max(a,a) = a

- *Essentially-unary* k-ary operation maps $(x_1,x_2,\ldots,x_k)$ to $f(x_i)$ for some $f(x)$ and i. A projection is essentially-unary k-ary operation with $f(x) = x$. We essentially want to map a collection of items to only one item. For example:
  - $\pi_b(a,b) = b$.
- *Semi-projection* k-ary operation maps $(x_1,x_2,\ldots,x_k)$ to x_i in only some cases. It requires that $k\geq3$ and its examples are contrived.
- *Majority* operation maps (a,b,c) to its most common element.
- *Affine* operation maps (a,b,c) to $a+b^{-1}+c$, where $\langle D,+\rangle$ is an Abelian group.
- An *Abelian* group is a pair $\langle D,+\rangle$
  - D is a domain that contains an identity element i: a + i = a. In D, every element a has an inverse a^{-1} under +: $a + a^{-1} = i$.
  - + is an operation on D^2 such that D is closed under +, and + is associative and commutative.

Let s be a solution to $U_k(Q)$, the k-ary operation $\hat{s}$ associated with s is defined as:

- The value of $\hat{s}(x)$ is the value assigned to the variable with name x in the solution s. For example, given $Q = \{A\oplus B,\neg A\}$, there exists a solution to $U_3(Q)$ (Figure 7), where v111 = 1; v110 = 1; v101 = 1; v100 = 0; v011 = 1; v010 = 0; v001 = 0; v000 = 0.
  - Then $\hat{s}(1,1,1) = 1$, $\hat{s}(1,0,1) = 1$, $\hat{s}(0,1,0) = 0$, etc.

Complexity of constraint languages: Theorem 11.4

Assuming P≠NP, any tractable constraint language over a finite domain must have a solution to its universal gadget associated with either a constant operation, a majority operation, an idempotent binary operation, an affine operation, or, a semi-projection. This is necessary but not sufficient.

- From the previous example, we get that $\hat{s}(1,1,1) = \hat{s}(1,1,0) = \hat{s}(1,0,1) = \hat{s}(0,1,1) = 1$ $\hat{s}(1,0,0) = \hat{s}(0,1,0) = \hat{s}(0,0,1) = \hat{s}(0,0,0) = 0$. This implies that $\hat{s}$ is a majority operation (because the values are major in the tuples), and as such Q is not tractable.
- If all solutions to $U_{|D|}(Q)$ are essentially-unary, then Q is NP-complete (corollary).
  - $U_{|D|}(Q)$ has ${}^{2}|D| = |D|^{|D|}$ variables (tetration), and combinatorial in the number of constraints.
  - Finding an exhaustive solution in this case will be very tedious.

Complexity of constraint languages: Sufficient conditions for tractability

Some sufficient conditions for a tractable constraint language depends on the following definitions:

- A relation R *allows* an operation *w*, if *w* is associated with a solution to $U_k(\{R\})$. We essentially want to take a solution and read it off element by element.
- *Inv(w)* is the set of R such that R allows *w*.
- An example of constant operations:
  - Let w be any constant operation on D: w(x) = C for all x.
  - Let Q=inv(w): all relations have only one support.
  - Q is always tractable because each constraint determines all variables in its scope, and if two constraints disagree, no solution exists (either a constraint is satisfied or not).
  - The constant language is categorized under this example.
- An example of semi-lattice operations:

    - Let + be any operation which is idempotent, commutative, and associative: x+x = x; x+y = y+x; (x+y)+z = x+(y+z)
    - Let Q=inv(+)
    - Q is always tractable and the following procedure finds a solution:
        - Establish GAC on the problem.
        - Is any domain is empty (after GAC), no solution exists.
        - Otherwise, return the solution where for each variable *v* with domain $d=\{d_1,d_2,\dots,d_k\}$ the value of *v* is $d_1+d_2+\dots+d_k$.
    - Horn-SAT is categorized under this example.
- An example of near-unanimity operations:
    - Let *w* be any k-ary operations which requires near-unanimity, i.e., all arguments but one must agree (it returns the most common argument): the 3-majority operation: w(x,x,y) = w(x,y,x) = w(y,x,x) = x for all x,y
    - 2-SAT is categorized under this example.
- The basic idea of the examples is that even though the specifics of constraint languages vary significantly, the operations associated with solutions to their universal gadgets determine quite effectively whether or not a given language is tractable.
- The necessary and sufficient conditions for tractability are unknown for most cases.

Hybridization of constraint languages

- *Relational subclasses*: specific sets of CSPs are determined by their constraint languages.
- *Structural subclasses*: specific sets of CSPs are determined by the properties of their hypergraphs (tree-structure, clique subgraphs)
- *Hybrid subclasses*: specific sets of CSPs are determined by their constraint languages and the properties of their hypergraphs.
- There are no particular heuristics for the tractability of hybrid subclasses.

Hybridization of constraint languages: Example

Given any constraint problem C with domain size d and maximum constraint arity r, then if C is strong d(r+1)-consistent, it is globally consistent.

- This subclass is tractable.
- It is dependent on the language (domain size and constraint arity).
- It is dependent on the structure (consistency requirements).