Hybrid Reasoning for Complex Systems

Mark Boddy and **Kurt Krebsbach**
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{boddy,krebsbac}@htc.honeywell.com

Abstract

We have recently begun work on extending our least-commitment constraint-based scheduling technology to handle more complex dynamical models. The current scheduling process involves a complex interaction between discrete choices (resource assignments, sequencing decisions), and a continuous temporal model. Discrete choices enforce new constraints on the temporal model. Those constraints may be inconsistent with the current model, thus forcing backtracking in the discrete domain, or if consistent, may serve to constrain choices for discrete variables not yet assigned. The extensions currently underway involve adapting and modifying more complex continuous models. This will permit us to apply the system to such complex problems as crude blending or tank management at a refinery, aircraft mission planning, or spacecraft mission planning and control.

1 Introduction

Accurate analysis and effective control of physical systems involving complex interactions between a continuous dynamical system and a set of discrete decisions is a common need in a wide variety of application domains. Effective design, simulation and control of such *hybrid systems* requires the ability to represent and manipulate models including both discrete and continuous components, with some interaction between those components.

For example, operating a petroleum refinery involves a set of discrete decisions regarding how much to make of what products, in what order, and where to store them. Economical production of those products involves controlling a complex process best modeled as a set of nonlinear equations and inequalities over a continuous, n-dimensional space. Control inputs and production decisions interact in complex ways. Certain production decisions (for example the manufacture of very small batches of product with rapid changeovers) are ruled out, based on the fact that the resulting plant operation would be badly suboptimal. Control inputs are in turn affected by production decisions: the plant may be run slightly sub-optimally in order to meet a production deadline for a valued customer. In addition to refinery operations, hybrid systems appear in domains including batch manufacturing, satellite and spacecraft operations, and transportation and logistics planning.

Traditionally, the continuous and discrete aspects of hybrid systems have been addressed separately. Typically, the discrete problem is solved using simplifying assumptions about the continuous behavior of the resulting system operation. During execution, operators are faced with a choice between optimal operation, tending to diverge rapidly from the predicted schedule, and controlling the system to synchronize with the behavior assumed in the discrete solution, resulting in substantially suboptimal behavior. This lock-step, sequential control structure is both limiting and unnecessary. Recent progress in both artificial intelligence (AI) and operations research (OR) has provided a set of tools and techniques for constructing integrated solutions in which the interactions between discrete and continuous control decisions can be handled much more flexibly, resulting in more effective solution methods and improved system operation.

In this paper, we describe a program of research, currently underway, aimed at developing an integrated framework for hybrid systems. This work builds on previous work in the area of constraint-based scheduling, in which we explicitly construct and manipulate hybrid models. Two specific advances we expect from the current work are 1) the ability to handle hybrid systems involving complex dynamic models and 2) a control structure allowing nontrivial problem-solving and decision-making within the continuous part of the problem.

2 Context and related work

The work we are doing is more synthesis than boldly striking off in a brand new direction. Essentially, it is a matter of taking an existing approach to hybrid systems (constraint envelope scheduling, or CES), and extending it by adopting results from elsewhere, with some necessary adaptation as well. The previous work on which we build comes predominantly from the fields of Artificial Intelligence (AI) and Operations Research (OR), and within those fields from that work addressing the solution of constraint satisfaction and constrained-optimization problems. In this section, we summarize these previous results. In the next, we describe in more detail both our synthesis and the extensions we have made or intend to make.

2.1 AI and OR

The modeling and solution of constraint satisfaction and constrained optimization problems using continuous or discrete models is the subject of active research in both AI and OR. The difference in the work done within these communities is more a matter of emphasis than of fundamental methodological distinctions. Broadly speaking, the

OR community has taken a more mathematical approach, stressing theoretical and empirical analysis of models expressed as systems of equations and inequalities. This emphasis has led to strengths in terms of the scale of the problems that can be solved, the generality of the resulting tools, and a certain degree of precision in terms of how well a given approach will do on a specified class of problems. Particularly in the continuous domains, OR approaches are very strong in being able to "bound" a problem, in the sense of determining whether or not a solution exists, given the current set of constraints upon that solution.

Work in Artificial Intelligence, and particularly within the Constraint Logic Programming community, has for the most part addressed the effective representation and solution of problems involving discrete models. This difference in focus has historical roots, including the influence of early (and ongoing) work on theorem-proving and propositional satisfiability, and the traditional AI focus on constructing complex models for "real world" problems. For these reasons, work on constraint satisfaction problems (CSPs) in AI has focussed predominantly on discrete problems, with a greater emphasis on exploiting the structure and semantics of the model in heuristics used to guide search. A related thread has been the development of "pruning" strategies, which are in some sense the complement of heuristics: rather than identifying promising paths to explore, they are used to conclude that some part of the search space need not be examined.

It is only recently that researchers in both AI and OR have made systematic efforts to integrate results from these two disciplines. [6] What they are discovering is that there is a considerable degree of commonality in the approaches taken, and an even greater degree of synergy. For example, van Hentenryck's HELIOS [10] system for general-purpose constraint solving and global optimization in continuous domains is based on a branch and prune algorithm, combining techniques from AI and numerical analysis, which has also been extended into a branch and bound algorithm for solving global optimization problems. This approach borrows both from advanced numerical methods in operations research and from search techniques for handling disjunction developed in the constraint logic programming community.

2.2 CSPs and COPs

The statement of a constraint satisfaction problem (CSP) includes a set of variables $V = \{v_1, v_2, \ldots, v_n\}$, taking on values from a set of domains $D = \{d_1, d_2, \ldots, d_n\}$, and a set of constraints C. The domains in D may be discrete or continuous. The elements of C are relations on the domains in D, specifying allowable combinations of values for the variables in V. The typical form of "solution" for a constraint satisfaction problem is to find an assignment to all of the variables in V, drawn from the domains in D, such that all of the constraints in C are satisfied, or to determine that no such solution exists. Constrained optimization problems (COPs) add to this an objective function or preference relation over the possible assignments to V. The problem is then to find a maximal (most preferred) assignment to V that satisfies C.

Constraint satisfaction and constrained optimization problems in either continuous or discrete domains can be stated in the same way. Given a set of variables, the objective is to find a corresponding set of assignments (equivalently, to bound the set of possible assignments), such that all constraints are satisfied. Solving such problems in either domain involves the same small set of general techniques: *propagation* of constraints so as to reduce variable domains, *bounding* to determine whether an acceptable solution lies within the set of assignments consistent with the current set of constraints, and the explicit addition of constraints on variable values so as to explore different parts of the current space of possible assignments (i.e., *search*). In general, OR has done more work on bounding and continuous domains, AI more on propagation and discrete domains. Both communities have made contributions to techniques for search. Recent integrations of AI and OR techniques have combined propagation and bounding techniques for both discrete domains (e.g., job shop scheduling) and continuous domains (e.g., modeling and control of chemical reaction kinematics).

2.3 Discrete Models

A *discrete* CSP or COP is simply one in which the domains in D are discrete. Discrete optimization problems can be formulated as either constraint satisfaction problems (CSPs) or as integer linear programming (ILP) problems. Although ILP methods appear to be more powerful, sometimes constraint programming can solve these problems more quickly, or even in some cases solve problems that could not be solved with an ILP formulation [12].

Work on solving discrete CSPs in AI has led to advances in the areas of search heuristics, search control, pruning and propagation methods, and static analysis of problem structure. Expressive domain models are better able to capture the manifold (and significant) details of complex problem domains. This extra information can then be used to guide problem-solving behavior, for example in the use of highly tuned, domain dependent heuristics. OR work on constant-approximation algorithms [8] provide a way of improving the "bound" computed for branch-and-bound search for optimal or near-optimal solutions of discrete problems.

Integration of AI and OR techniques in the solution of discrete CSPs includes recent work on scheduling problems [1], as well as more abstract theoretical investigations of the combination of search, propagation, and bounding methods drawn from both communities [11].

2.4 Continuous Models

A *continuous* CSP or COP is one in which the domains in D are dense, for example drawn from subsets of the real numbers. Variable "assignments" bounded by intervals which are "small enough" according to some solution criterion, typically input by the user. To date, continuous CSPs have received a great deal of attention in OR, and rather little in AI. There is a wide variety of commercially-available solvers for continuous problems, some of them quite sophisticated. Available systems are capable of monitoring the numerical conditioning of the problem, handling large-sparse problem spaces, restarting given partial solutions, of providing sensitivity information about the shape of the solution found, or of providing information about their inability to find a feasible solution. These systems include linear programming solvers such as the venerable XMP, or LINDO, or CPLEX (using the new interior point methods), and nonlinear solvers such as MINOS, CONOPT, NPSOLV, LANCELOT, and SQP. Over 100 such systems are currently available.

Van Hentenryck's HELIOS system, cited above, integrates methods drawn from both AI and OR in the solution of continuous-domain constraint satisfaction and optimization problems. HELIOS is based on a branch and prune algorithm [9], combining techniques from AI and numerical analysis, which has also been extended into a branch and bound algorithm for solving global optimization problems. Pruning is accomplished using *box consistency*, a continuous-domain analogue of arc consistency, a well known concept in artificial intelligence. Helios combines box-consistency for pruning and propagation with more traditional numerical methods for bounding and a "splitting" operator to explore complex solution spaces in an efficient manner. This approach borrows both from advanced numerical methods in OR and from search techniques for handling disjunction developed in the constraint logic programming community.

2.5 Hybrid Models

The only substantial body of research addressing CSPs and COPs involving both continuous and discrete variable domains has been in Operations Research, in the area of Mixed-Integer Linear Programming. For the most part, this work enjoys the benefits and suffers the disadvantages of related OR approaches in linear programming. One additional problem is related to the strictures of the mathematical model imposed: there is frequently a combinatorial "blow-up" in the size of the model for what was initially a fairly small problem. For example, representing a simple batch manufacturing problem as an MILP can result in upwards of 20,000 variables, for a problem which started with 20 machines, 25 batches to be processed, and a time granularity of only half an hour, over a duration of a week. Work in AI has not gone further, if anything not as far. Heterogeneous models (e.g., in scheduling domains) have been handled in an *ad hoc* manner, with little if any systematic analysis of the available design alternatives, much less the suitability of those alternatives to particular classes of problems.

3 Hybrid Systems and CES

In this section, we describe our existing scheduling capability, which embodies a specific approach to solving hybrid systems, and how we are in the process of extending it.

3.1 CES

Constraint envelope scheduling [2] is a "least-commitment" approach to constraint-based scheduling, in which restrictions to the schedule are made only as necessary. Our scheduling approach is based upon the following principles:

- User visibility into an evolving schedule. For problem-solvers (human or otherwise) to make intelligent decisions or evaluate scheduling choices made automatically, they must be able to determine the consequences of those decisions. We have developed inference methods, display techniques, and query mechanisms specifically designed to efficiently extract the maximum amount of useful information from a partially-specified schedule.
- Explicit representation of scheduling decisions (e.g., the decision to order two activities to remove a resource conflict). This permits identification of the source, motivation, and priority of scheduling decisions, when conflicts arise in the process of schedule construction or modification (e.g., due to negotiation between competing agencies, each with partial control over the scheduler).

The resulting approach supports a continuous range of scheduling approaches, from completely interactive to completely automatic. We have built systems across this entire range of interaction styles, for a variety of applications including aircraft avionics [3], image data analysis and retrieval [4], spacecraft operations scheduling, and discrete and batch manufacturing [7], among others, with problem sizes ranging up to 30,000 activities, and 140,000 constraints.

Explicitly modeling scheduling decisions as additional constraints facilitates the design and implementation of schedulers that support iterative schedule negotiation and rescheduling, incremental schedule modification, conflict identification and repair, and on-line, real-time rescheduling in an operational environment. As suggested by Smith [13], constraint-based schedulers can provide facilities for computer-supported collaborative work (CSCW): the schedule can function as a "blackboard" on which multiple users can post constraints and examine their effects and interactions.

3.2 Solving hybrid models in CES

Implementing scheduling systems has provided us with both a set of test problems and the motivation for some initial steps towards the solution of heterogeneous problems. Any scheduling problem more complex than a simple job shop (for example, one that also includes the possibility of assignable resources) is best modeled as a heterogeneous problem.

In the current CES approach, the continuous-domain solver, capable only of representing linear inequalities involving at most two variables, functions as a subroutine of the discrete solver, reporting inconsistencies and responding to queries regarding the pruning of discrete variable domains. We represent and manipulate hybrid models using two key technologies: a flexible search engine (the discrete-side, search-based problem solver), and the *Interval Constraint Engine* (ICE), a constraint-based temporal inference engine.

In the process of developing and applying CES, we have discovered that one of the keys to providing an integrated solver for hybrid models is the proper handling of the tradeoffs and interactions between the discrete and continuous domains. For example, explicit non-convex disjunction in the continuous domain makes bounding inferences ineffective. Moving this disjunction from the continuous model to an implicit representation in the discrete model enables more effective bounding and propagation in the continuous domain, at the expense of serial rather than parallel exploration of the problem space.

Among the other techniques we have adopted or are considering adopting as part of CES are OR approaches to both pruning and bounding. The pruning method is known as "edge-finding," and leads to the elimination of additional ordering choices based on those scheduling decisions made so far [5]. The bounding method is a fairly standard "energy" calculation, involving summing the duration for a set of unordered activities to determine whether there is any feasible ordering.

Use of these techniques in an incremental, repair-based search control structure depends on many of the same properties we expect to be relevant for a fully general integration of continuous and discrete problem-solving. The continuous representation of the problem (in ICE) is incrementally updatable for both addition and deletion of constraints, sound and complete with respect to the detection of inconsistencies, and includes hooks permitting the system (or the user) to determine the correspondence between the continuous-domain constraints involved in an inconsistency and the discrete-domain variable assignments that led to those constraints. In addition, specialized inferences cached in ICE permit the system to prune discrete variable values (e.g., activity ordering choices), based on continuous-domain constraints.

For hybrid systems in which the continuous model is more sophisticated (e.g., refinery scheduling incorporating process optimization), this approach is insufficient—the continuous solver must have more expressive power, and must be more fully integrated into the overall problem-solving control structure. In addition to the expressive weakness of ICE, the current system is limited in that the continuous-domain solver is treated as a subroutine of the discrete solver, reporting inconsistencies and responding to queries regarding pruning of discrete variable domains, rather than being fully integrated into the overall problem-solving control structure. In the next section, we discuss how we plan to build on these initial steps in the investigation of fully-integrated problem-solvers for hybrid models

4 Extensions

There are three main areas to look at: the continuous solver, the discrete solver, and the control structure that manages the interaction between them. In particular, we seek answers to the following questions:

- What is the space of (useful) control strategies? How are choices in the design of these strategies related to features of the problem being solved? What kinds of hooks are necessary in the discrete and continuous solvers being integrated for a particular control strategy to be feasible? What kinds of control structures are best suited to maintaining a least-commitment, incremental solution methodology?
- What kinds of tradeoffs are appropriate, between inferential efficiency in the continuous model and expanding the discrete model to make disjunctions explicit? What kinds of disjunction can be added to the continuous model? How do modeling choices affect these tradeoffs?
- What kind of search engine is appropriate, for a given set of solvers and control structure?

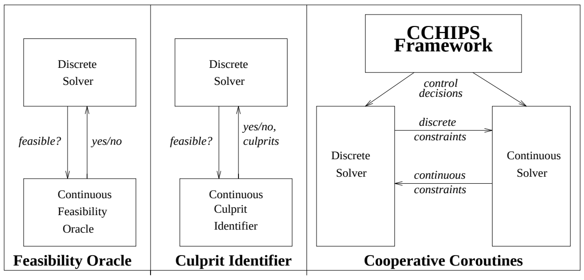

Figure 1: Three different integration strategies

The last of these three questions is one whose answer will be empirically determined, and which is very likely to vary from application to application. Consequently, we currently have little to say about the issue that goes beyond relating past experience, and stating an intension to extend the system in such a way as to make it easy to experiment with various search strategies.

Our approach to addressing the first two questions comprises the remainder of this section.

4.1 Process Extensions

We propose to extend the current CES architecture to more complex hybrid systems. This extension will require the integration of our current discrete solution methods with a more complex continuous domain solver, probably an NLP solver, within a control architecture that grants more autonomy and responsibility to the continuous solver.

There is a wide spectrum of possible strategies for problem-solving control for hybrid problems. Three representative points on that spectrum, illustrated in Figure 1, illustrate the range of possibilities. The simplest architecture is the *Feasibility Oracle*, in which there is a single solver, interacting with a second subsystem responsible for checking the current partial solution for feasibility. This architecture is not compatible with the implementation of flexible search strategies designed to focus on specific problem hot-spots or bottlenecks, nor with local search or repair-based methods for incremental resolution given an existing assignment, an important capability for real applications. The *Culprit Identifier* is a slightly more complicated control architecture, in which the subsystem monitoring feasibility also returns information about *why* the model is inconsistent, i.e., a set of constraints which are mutually inconsistent. Given culprit identification, the solver has the information necessary to implement a variety of conflict resolution strategies involving reassignment, variable reordering, or constraint relaxation. The current CES architecture uses a culprit identification approach. Finally, *Cooperative Coroutines* is a control architecture intended for use when both the continuous and discrete aspects of the problem require nontrivial problem-solving effort. Decisions may be made in one domain or the other, rather than only in one of the two, with effects propagating in both directions and ultimately through one or more cycles between the solvers.

Which of these control structures can be applied depends on the capabilities of the individual solvers. A solver applied as a "feasibility oracle" is likely to need no modification at all—any of the LP solvers listed above could be used in this mode without modification. Culprit identification is trickier; it is necessary to be able to identify the specific decisions or constraints involved in an infeasibility, and to somehow annotate those constraints in a way understandable by the controlling solver (i.e., so that the variable assignments leading to the infeasibility can be identified).

Cooperative coroutining is the most flexible form of interaction, and the one making the most demands on the individual solvers. A systematic investigation of the requirements for cooperative coroutining has yet to be done, but previous experience with culprit identification in CES provides us with some guidance, in that cooperative coroutining is essentially a symmetric form of culprit identification with some added control structure. Accordingly, we can conclude that solvers involved in cooperative coroutining will need at least to be incremental, in the sense that (sets of) constraints can be added or deleted, with efficient checks for consistency and other inference. Infeasibility or inconsistency must be detectable, and culprit sets as described above identified to be passed either to the complementary solver or to the overall control structure.

These requirements embody the central risk involved in this project. We have argued, first, that cooperative coroutining is needed for the solution of complex hybrid systems, and second, that those same problems required the use of a sophisticated continuous-domain solver. It is possible that we will not be able to find a solver of sufficient power which either itself satisfies the requirements or which can be wrapped in an insulating layer within which these requirements can be satisfied.

4.2 Expressive Extensions

The desired results of this research program include an architecture within which specialized solvers for continuous and discrete domains can be integrated with one another in a "mix-and-match" approach to providing tailored solutions for particular applications. The restrictions placed on choices for individual solvers for the discrete and continuous domains will depend strongly on the control structure. A solver applied as a "feasibility oracle" is likely to need no modification at all—any of the LP solvers listed above could be used in this mode without modification. Culprit identification is trickier; it is necessary to be able to identify the specific constraints involved in an infeasibility, and to somehow annotate those constraints in a way understandable by the controlling solver (i.e., so that the variable assignments leading to the infeasibility can be identified).

The appropriate handling of disjunction is an additional design issue. In both continuous and discrete domains, disjunction can either be represented in the domain model (e.g., using disjunctive temporal constraints), present in the set of variables to be assigned (a variable for activity ordering), or implicit in search control for the problem solution (at some point in the search, an ordering decision is made). These choices have significant consequences: disjunction represented explicitly in the model can frequently be used to further bound the problem, but makes inference more expensive. The fact that these inferences must be made repeatedly as the search for a solution progresses places strong constraints on how complex the inference can be. Instead of limiting the expressive capabilities of the model, it is sometimes possible to perform *approximate* inference on a more expressive model, thus preserving the necessary efficiency at the cost of getting answers that are correct, but may not be complete.

5 Summary and Conclusions

The work described here is preliminary, in the sense that our existing capability to represent and solve hybrid constraint problems has undesirable expressive restrictions that we are actively working to remove. Success in this endeavor will result in the extension of the benefits of the CES scheduling technology to more complex hybrid systems, in particular those involving a nontrivial continuous optimization problem. These benefits include incremental (re)solution of an existing model, flexible problem-solving focussed through identification of conflicting constraints and decisions, and mixed-initiative interaction with a more general (possibly a human) problem-solver.

Despite the existence of active research communities in several relevant areas (discrete and continuous optimization, and the synthesis and analysis of hybrid control systems, e.g.), the research literature does not provide significant guidance to us in making these extensions, though there are hints here and there as to which approaches may be most fruitful. But the effective treatment of complex hybrid systems appears to be a significant, and so far largely unaddressed, area for further research.

References

[1] Baptiste, P., Le Pape, C., and Nujitem, W., Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling, *Working Notes of the Joint Workshop on Artificial Intelligence and Operations Research*, Timberline, OR, 1995.

[2] Boddy, Mark, Carciofini, Jim, and Hadden, George, Scheduling with Partial Orders and a Causal Model, *Proceedings of the Space Applications and Research Workshop*, Johnson Space Flight Center, August 1992.

[3] Boddy, Mark and Goldman, Robert, Empirical Results on Scheduling and Dynamic Backtracking, *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation for Space, Pasadena, CA*, 1994.

[4] Boddy, Mark, White, Jim, Goldman, Robert, and Short, Nick, Integrated Planning and Scheduling for Earth Science Data Processing, Hostetter, Carl F., (Ed.), *Proceedings of the 1995 Goddard Conference on Space Applications of Artificial Intelligence and Emerging Information Technologies*, Available as NASA Conference Publication 3296, 1995, 91–101.

[5] Carlier, Jacques and Pinson, Eric, A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem, *Annals of Operations Research*, **26** (1990) 269–287.

[6] Ginsberg, Matt, *Working Notes of the Joint Workshop on Artificial Intelligence and Operations Research*, (Timberline, OR, 1995).

[7] Goldman, Robert P. and Boddy, Mark S. Constraint-Based Scheduling for Batch Manufacturing, *IEEE Expert*, (1997).

[8] Hall, Leslie L., Shmoys, David B., and Wein, Joel, Scheduling to Minimize Average Completion Time: Off-line and On-line Algorithms, *ACM-SIAM Symposium on Discrete Algorithms (SODA '96)*, Atlanta, GA, 1996.

[9] Hentenryck, P. Van, McAllister, D., and Kapur, D., Solving Polynomial Systems Using a Branch and Prune Approach, *SIAM Journal on Numerical Analysis*, (1995).

[10] Hentenryck, Pascal Van and Michel, Laurent, *Helios: A Modeling Language for Nonlinear Constraint Solving and Global Optimization using Interval Analysis*, Technical Report CS-95-33, Department of Computer Science, Brown University, 1995.

[11] Rodosek, Robert, Combining Constraint Network and Causal Theory to Solve Scheduling Problems from a CSP Perspective, *ECAI-94*, 1994, 630–634.

[12] Smith, B., Brailsford, S.C., Hubbard, P.M., and Williams, H.P., The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared, *Working Notes of the Joint Workshop on Artificial Intelligence and Operations Research*, Timberline, OR, 1995.

[13] Smith, Steven F., Mixed-Initiative White Paper Notes, archived at ISX.