A Framework for Dynamic Constraint Reasoning using Procedural Constraints

Ari K. Jónsson
NASA Ames Research Center
Mailstop 269-2
Moffett Field, CA 94035
jonsson@ptolemy.arc.nasa.gov

Jeremy D. Frank
NASA Ames Research Center
Mailstop 269-3
Moffett Field, CA 94035
frank@ptolemy.arc.nasa.gov

Abstract

Many complex real-world decision and control problems contain an underlying constraint reasoning problem. This is particularly evident in a recently developed approach to planning, where almost all planning decisions are represented by constrained variables. This translates a significant part of the planning problem into a constraint network whose consistency determines the validity of the plan candidate.

Since higher-level choices about control actions can add or remove variables and constraints, the underlying constraint network is invariably highly dynamic. Arbitrary domain-dependent constraints may be added to the constraint network and the constraint reasoning mechanism must be able to handle such constraints effectively. Additionally, real problems often require handling constraints over continuous variables. These requirements present a number of significant challenges for a constraint reasoning mechanism.

In this paper, we introduce a general framework for handling dynamic constraint networks with real-valued variables, by using procedures to represent and effectively reason about general constraints. The framework is based on a sound theoretical foundation, and can be proven to be sound and complete under well-defined conditions. Furthermore, the framework provides hybrid reasoning capabilities, as alternative solution methods like mathematical programming can be incorporated into the framework, in the form of procedures.

1 Introduction

Constraint reasoning has been proven to be an effective technique for representing and reasoning about a variety of real-world decision problems. This includes problems in reactive and deliberative autonomous control. The interest in autonomous systems for control has grown steadily over the last few years, particularly in applications where human intervention is impossible, time-consuming or costly. Spacecraft, rovers and other remote artifacts are excellent examples of such applications. Consider a spacecraft that is coasting through space, on its way to a destination. Due to the limited availability of communication facilities and planetary and stellar bodies blocking the spacecraft's communication lines to Earth, human intervention is impossible for long periods of time. Even when humans can intervene, the time required to determine the state of the spacecraft, formulate a control sequence, test it, and then execute it, is typically measured in hours if not days. Finally, having engineers, spacecraft systems specialists and other necessary staff constantly available to control the spacecraft, is a significant portion of the overall mission cost.

The need to reduce spacecraft mission costs, in conjunction with the desire to explore such faraway places as Europa and Pluto, has driven home the need for autonomous spacecraft control systems. Over the last few years, researchers have worked on developing a model-based autonomous control architecture that can be adapted to different spacecraft and missions. This effort produced the Remote Agent architecture, a complete autonomous control system for spacecraft. It also produced a complete implementation that successfully controlled the Deep Space One spacecraft for two days in May of 1999, as part of the Remote Agent Experiment [MNPW98].

The Remote Agent has three main components:

- A planning and scheduling component that builds detailed plans from high-level mission goals. These plans take into account resources, temporal information, configuration constraints and other aspects of controlling a spacecraft.

- An intelligent execution component that executes the plan, by performing the specified tasks and maintaining the prescribed conditions.

- A mode identification and recovery component that compares the results of executed commands with a model of the spacecraft, in order to detect and analyze any faults. In addition to identifying likely faults, it can generate alternative configurations to get around a fault.

Constraint reasoning plays an important role in all parts of the Remote Agent (RA), but the most involved constraint reasoning is done in the planner. The reason is that the RA planning approach utilizes constrained variables to specify action instances, maintenance conditions, temporal information, and various other aspects of a complex plan. This makes constraint reasoning a significant part of the planning process. However, the planning task imposes a number of requirements on the underlying constraint reasoning system: higher-level decisions alter the set of variables and constraints, arbitrary constraints must be effectively dealt with, and real-valued variables must be represented.

Our work is aimed at providing a general framework for the constraint reasoning capabilities that are necessary for autonomous control applications like the planner. This particular work is part of a larger effort to define and build the next generation RA planning system [JMMR99], which provides a clearer and more powerful definition for the planning framework and increases the generality and efficiency of the resulting planner system. Our work has resulted in a general constraint reasoning framework that provides all the functionality needed for the new RA planner. A prototype of the framework has been implemented and integrated into the new system. Nonetheless, the framework is independent of the planning system and can be utilized in other applications where dynamic constraint reasoning is needed.

We begin by giving a brief introduction to the new RA planner in the next section. We then go on to give a quick overview of the concepts in dynamic constraint reasoning, before turning our attention to the concepts in procedural reasoning in constraint satisfaction. In the following section, we present the overall framework, including the handling of real-valued variables and the use of hybrid reasoning. We conclude by reviewing our accomplishments and how this work will progress in the future.

2 New Remote Agent Planner

The RA planning framework is based on the Heuristic Scheduling Testbed System (HSTS) [Mus94]. In both systems, complex actions and parameterized fluents are described in a unified manner. The world is described with *state variables* (also called *timelines*), whose values change over time. The actions and fluents are then uniformly described as *segments* where a state variable maintains the appropriate state. Each segment is described using a set of CSP variables. The temporal aspect of a segment is described using variables that represent the start time, the end time, and the duration of the segment. The remaining aspect, the state value described by the segment, is also described by a set of CSP variables. This is because actions can have parameters that instantiate the action, e.g, a thrusting action will have direction and thrust level parameters. *Planning axioms* specify relations between segments, enforcing preconditions, effects, enabling conditions, mutual exclusions, etc. These axioms then give rise to constraints between the different CSP variables that describe the different segments. A set of connected segments, i.e, a candidate plan, therefore gives rise to an instance of a constraint network. In order for the candidate plan to be valid, there must be a valid solution to the underlying constraint problem.

As planning decisions are made, the constraint network changes, to represent new segments and eliminate variables stemming from old segments. As a result, the constraint reasoning must be able to efficiently process changes to the constraint network, and should also be able to efficiently respond to queries about a sequence of closely related constraint problems.

The new RA Planner is a general planning framework which uses a *model* to specify the domain in which the planning is to take place. The model consists of descriptions of the state variables and the planning axioms. This has the advantage that the same system can be applied in different situations, from fully autonomous on-board spacecraft control to on-ground interactive planning and scheduling. Consequently, the constraint reasoning component must be able to efficiently represent and reason about arbitrary constraints. The use of exhaustive enumeration of tuples for arbitrary constraints, while theoretically useful, is too inefficient for such applications.

Finally, any realistic spacecraft domain, like most other real-world domains, has components which must be modeled using continuous variables. As a result, the RA planner must be capable of representing and reasoning about continuous variables, which in turn requires the constraint component to be able to handle them. Although many constraints over continuous variables have no closed solutions, it is still crucial that the constraint reasoning component be able to handle real-value variables in a well-behaved and yet effective manner.

3 Dynamic Constraint Reasoning

Conceptually, a constraint satisfaction problem is a set of variables, each of which must be assigned a value from a given domain, and a set of constraints, each of which limits the set of allowed combination of variable assignments. Formally, a *Constraint Satisfaction Problem* or CSP is a triple $P = (V, D, C)$, where:

1. $V = \{v_1, \ldots, v_n\}$ is a set of variables
2. $D = D_{v_i} \mid i \in \{1, \ldots, n\}$ are the domains of the variables, where each D_{v_i} is a finite set of possible values of v_i.
3. C is a set of constraints (Y_j, R_j), where each constraint consists of a scope $Y_j = \{v_{i_1}, \ldots, v_{i_k}\} \subseteq V$ and a relation $R_j \subseteq \prod_{\nu=1}^{k} D_{v_{i_\nu}}$.

A *valid solution* to a constraint satisfaction problem $P = (V, D, C)$, where $V = \{x_1, \ldots, x_n\}$, is an n-tuple $(v_{x_1}, \ldots, v_{x_n})$, such that:

1. $v_{x_k} \in D_{x_k}$ for $k = 1, \ldots, n$, and
2. For any $(Y, R) \in C$ with $Y = \{x_{i_1}, \ldots, x_{i_k}\}$, we have $(v_{x_{i_1}}, \ldots, v_{x_{i_k}}) \in R$.

A constraint problem that has at least one solution is called *consistent.* Constraint problems that have no solutions are called *inconsistent.*

In the spacecraft planning domain, where control involves searching alternate plans, the underlying constraint problem changes as different planning decisions are made. However, each problem is closely related to the previous one, making it more effective to view the constraint problem as a dynamic problem, rather than as single instances of static problems.

To formalize the notion, let $P = (V, D, C)$ be a constraint satisfaction problem. Any problem of the form $Q = (V', D', C')$ such that $V' \supseteq V$ (i.e. there are more variables), $D'_v \subseteq D_v$ for each $v \in V$ (i.e. there are fewer legal values for variables) and $C' \subseteq C$, (i.e. there are fewer legal combinations for variables in a constraint) is a *restriction* of P. Any problem of the form $Q = (V', D', C')$ such that $V' \subseteq V$ (i.e. there are fewer variables), $D'_v \supseteq D_v$ for each $v \in V$ (i.e.there are more values for variables) and $C' \supseteq C$ (i.e. there are more legal combinations for variables in a constraint), is a *relaxation* of P. A *Dynamic Constraint Satisfaction Problem* or DCSP is a sequence of constraint satisfaction problems $P_0, P_1, \ldots$, such that each problem P_i is either a *restriction* or a *relaxation* of P_{i-1}. This definition is consistent with similar definitions given in [Dec88].

We are typically interested in the consistency of each constraint problem instance. The reason is that if the variables describing each segment cannot be instantiated without violating constraints, then any plan which gives rise to the network is invalid.

4 Procedural Reasoning

Whereas many applications of dynamic constraint reasoning are for specific domains, model-based control systems, like the new RA Planner, must be capable of handling differing domains with different constraints. This means that a general dynamic constraint framework for such applications must allow arbitrary constraints to be specified, and must be capable of handling them effectively. At the same time, efficient constraint reasoning is essential for any autonomous control applications. To satisfy these requirements, we look towards the concept of procedural reasoning, which has recently been formalized in the context of constraint satisfaction [Jón97]. The procedures defined by this work were initially developed to make constraint reasoning more effective, but procedures are also useful for specifying and using arbitrary constraints. We now give a brief overview of procedures and some of the related concepts and results. This is not intended as a complete overview of procedures; the details can be found in [Jón97].

The motivation behind procedures is to utilize specialized methods to solve certain subproblems, e.g, use suitable algorithms to solve arithmetic equations directly. Based on this idea, an *extension procedure* is defined as a function e that maps a given partial assignment p to another assignment $e(p)$, such that $p \subseteq e(p)$[1]. This definition is suitable for search techniques aimed at solving static constraint satisfaction problems, and turns out to have a number of useful properties for a wide range of search engines. For the purposes of specifying constraints for our dynamic constraint reasoning framework, a more powerful definition is used. An *elimination procedure* is a function e that maps a tuple of domains $\langle d_1, \ldots, d_k \rangle$, into another tuple $\langle d'_1, \ldots, d'_k \rangle$, such that each domain d'_i is a subset of the corresponding domain d_i. This stronger definition is used, because our constraint reasoning is aimed at enforcing certain levels of consistency, rather than just finding some solution.

It is fairly straightforward to see that under easily satisfied conditions, such an elimination procedure can be used to represent and enforce any constraint. The only necessary conditions are that the mapping e never eliminates a member of a domain that is a part of a solution, and that if each given domain is a singleton, then e maps the tuple into a tuple with an empty domain if and only if the singleton does not satisfy the given constraint. Just as importantly, the procedure can be implemented such that it enforces the constraint significantly more efficiently, both in time and space, than a declarative description of the allowed or disallowed variable assignments. As an example, an arithmetic constraint can be enforced efficiently enforced by direct calculations than by declarative axioms or exhaustive listing of valid solutions.

Both of these procedure definitions are special cases of *general procedures*, which map constraint networks and domain tuples into sets of additional constraints for the network. The reason we limit ourselves to elimination constraints in this particular framework is the cost associate with added constraints, and the unavoidable utility problem that follows.

Useful theoretical properties can be proved about certain classes of procedures and search engines, most notably regarding correctness, systematicity and completeness. The obvious subclass of procedures consists of those that are *correct*, meaning that no valid solutions are eliminated from consideration by applying the procedure. More formally, an elimination procedure e is correct, if for any domain tuple $\langle d_1, \ldots, d_k \rangle$, such that $\langle d'_1, \ldots, d'_k \rangle = e(\langle d_1, \ldots, d_k \rangle)$, the values in $d_i \setminus d'_i$ do not participate in any solution. Jónsson [Jón97] proved that, for a large class of complete and systematic algorithms, the use of any set of correct procedures will not affect completeness or systematicity. A larger, more interesting class is that of *weakly correct* extension procedures, which only guarantee that at least one solution will remain after the application of the procedure. For this class, Jónsson [Jón97] proved that search engines satisfying certain conditions will remain complete and systematic, even when arbitrary sets of weakly correct extension procedures are utilized. All well-known systematic algorithms, including non-chronological techniques like dynamic backtracking [Gin93], satisfy these conditions.

In many cases, the concept of decision variables can be used to eliminate variables and constraints from the problem, to reduce the problem size. The idea of only having to assign values to a subset of the variables to determine a consistency has been around for quite some time. For example, in satisfiability algorithms like Davis-Putnam [DP60], as soon as all clauses are satisfied, there is no need to instantiate remaining variables. The general concept is that of *decision variables*: given an elimination procedure e for a constraint satisfaction problem $C = (X, V, K)$, let $D \subseteq X$ be such that for any assignment to variables in D, applying e to that assignment results either in a problem that is globally consistent or in an empty domain. It is easy to see that it is sufficient to assign values to variables in D, as the procedure can be used to determine if the rest of the problem is solvable. The definition extends easily to a set of procedures, giving us a clear definition for decision variables for arbitrary sets of procedures. It should be noted that this definition significantly generalizes the concept of *control variables* in satisfiability problems, which is a set of variables from which the rest of the variables can be given values using unit propagation alone.

The formal concept of elimination procedures can be used to incorporate additional efficient reasoning

[1] The subset notation indicates that more variables have been assigned values in $e(p)$ than in p.

mechanisms into constraint reasoning techniques. These include continuous methods like solving linear programming subproblems, and discrete methods like determining earliest and latest start times for the variables in simple temporal networks. In essence, this formal framework provides a clear foundation for many types of hybrid reasoning in constraint satisfaction.

5 Framework for Dynamic Procedural Reasoning

The goal of this work is to utilize procedural reasoning as an approach to defining a framework for dynamic constraint reasoning. Let $(C_0, C_1, \ldots)$ be a dynamic constraint problem, such that each C_i is a restriction or relaxation of the previous problem C_{i-1}. In terms of our autonomous planning application, the changes from one problem instance to the next may be combinations of:

- Add or delete variables
- Add or delete constraints
- Modification of variable domains (including variable assignments being made or undone)

Each constraint network instance C_i is described in terms of a set of variables, each with a specific domain (unit domains represent assignments), and a set of constraints. There are two types of constraints, declarative and procedural. The declarative constraints are limited to a small set of constraints, handled directly by the dynamic constraint reasoning framework. In the planning applications, those are:

- Equality and inequality constraints
- Distance constraints for temporal variables[2]

All other constraints are described by elimination procedures. Since elimination procedures can be used to define arbitrary constraints, this allows the network to be applied to different domains with no changes except for adding the domain-dependent procedural constraints.

The framework allows for real-valued variables, as such variables are often necessary in domains such as spacecraft control. Unfortunately, sound and complete reasoning with arbitrary procedural constraints over continuous variables is not possible. We must therefore limit real variables to non-decision variables. Doing so guarantees that once a valid solution is found to the set of decision variables, which are all discrete, then the procedures can be used to determine the solution or inconsistency of the remaining variables. Note that the real-valued variables can still participate in constraints, and have an effect on the solvability of the problem. For an example, consider a real variable representing the remaining fuel after an engine burn. A constraint links the duration of the burn and the thrust level to the fuel-level variable, and if the thrust level is too high, or the duration too long, the constraint may indicate that the problem is unsolvable due to the remaining fuel being below the minimum.

The main role of the dynamic constraint reasoning framework is to respond to queries about each CSP. These queries include whether the CSP is consistent or not, which values in a domain may be eliminated from consideration, and so on. The framework allows for approximate answers, e.g, by only indicating that the instance has not been proven to be inconsistent, or by providing only a subset of the values that can provably be eliminated from a domain. This trades of effectiveness (providing useful results) against efficiency (cost of reasoning). The best known approaches to balancing this tradeoff are based on using limited consistency reasoning, like achieving arc-consistency and providing the approximate answers based on the results.

To further enhance the efficiency of the framework, it is possible to retain information derived by the reasoning process. For example, certain domain restrictions or assignments from correct elimination procedures can be reused for the next instance. The simplest sufficient condition for doing this is that the next instance is a restriction. A more sophisticated technique, based on storing the dependency information that led to a domain restriction, allows the reuse of some conclusions, even in the face of a relaxation.

Consider, for example, the constraint on spacecraft fuel consumption which we mentioned earlier. For any thrusting segment, the thrust level and segment duration affect the fuel consumption. Due to engine warm-up time and other complications, this relation is rarely linear. An elimination procedure enforcing this constraint can reduce the domains for the thrust level, duration and fuel consumption variables, based on current information. For example, if available fuel limits the fuel consumption, the procedure can limit the

[2]The temporal constraints form a simple temporal network which can be handled using efficient reasoning techniques, as described in [DMP91] and [TMM98].

thrust level and duration. It is easy to see that once the duration and thrust level have been determined, the fuel consumption can be calculated directly by this procedure. This makes the real-valued fuel consumption variable a non-decision variable.

6 Conclusions and Future Work

Our work is aimed at specifying a general, effective framework for performing constraint reasoning in model-based autonomous control applications. These applications demand that arbitrary constraints can be specified, that real-valued variables must be handled, and that the reasoning be efficient and effective. An example of such an application is the new RA Planner, which is being developed for future autonomous control applications on-board spacecraft and rovers.

To provide all the necessary functionality, we have defined a framework that uses elimination procedures to describe arbitrary constraints and to enhance the efficiency of the framework. The result is a well-defined framework that can be proven to be sound at all times, and complete with respect to fully determining consistency and the set of valid solutions. To retain efficiency, the framework also allows for incomplete but correct results, which can be found in much less time.

The framework has been implemented as a prototype that has been integrated into the new RA planner. The resulting constraint reasoning abilities cover all of the abilities needed for the original RA planner, which successfully flew on Deep Space One. As a result, we expect this framework to be useful in various dynamic constraint reasoning applications, including the next generation autonomous spacecraft control agents.

What is the cost of using constraint procedures over specialized procedures? For example, compare AC-x to using constraint procedures for arc consistency and see how much the overhead is.

Finding minimal sets of decision variables. Trading off complexity of decision procedures with number of decision variables.

References

[Dec88] A Dechter, R. & Dechter. Belief maintenance in dynamic constraint networks. *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 37–42, 1988.

[DMP91] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[DP60] M. Davis and H. Putman. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[Gin93] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[JMMR99] Ari K. Jónsson, Paul H. Morris, Nicola Muscettola, and Kanna Rajan. Next generation remote agent planner. In *Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS99)*, 1999.

[Jón97] Ari K. Jónsson. *Procedural Reasoning in Constraint Satisfaction*. PhD thesis, Stanford University, Stanford, CA, 1997.

[MNPW98] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian William. Remote agent: To boldly go where no ai system has gone before. *ai*, 103(1-2):5–48, August 1998.

[Mus94] N. Muscettola. Hsts: Integrated planning and scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*, pages 169–212. Morgan Kaufman, 1994.

[TMM98] Ioannis Tsamardinos, Nicola Muscettola, and Paul Morris. Fast transformation of temporal plans for efficient execution. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 254–261, Menlo Park, July 26–30 1998. AAAI Press.