A Framework for Dynamic Constraint Reasoning using Procedural Constraints

Ari K. Jónsson [1] and **Jeremy D. Frank** [2]

Abstract.
Many complex real-world decision problems, such as planning, contain an underlying constraint reasoning problem. The feasibility of a solution candidate then depends on the consistency of the associated constraint problem instance. The underlying constraint problems are invariably dynamic, as higher level decisions result in variables, values, and constraints being added and removed. In real-world reasoning applications, constraints may be arbitrarily complex, variables may have continuous domains, and neither variables nor values may be effectively enumerable beforehand. Such applications, therefore, present a number of significant challenges for a dynamic constraint reasoning mechanism.

In this paper, we introduce a general framework for representing and reasoning about dynamic constraint networks arising from complex real-world applications. It is based on the use of procedures to represent and effectively reason about general constraints. The framework can handle arbitrary changes to the network, including the addition and deletion of variables and values. It can reason with real-valued variables, and utilize special-purpose reasoning methods, such as arithmetic problem solving, in the form of procedures. The resulting framework is based on a sound theoretical foundation, which guarantees termination and correctness.

1 Introduction

Constraint reasoning has been proven to be an effective technique for representing and reasoning about a variety of real-world decision problems. Many important problems, including a variety of scheduling problems, have successfully been represented and solved as constraint satisfaction problems. However, static constraint problems are not sufficiently expressive for complex problem domains, such as real-world planning.[3] Such complex problems contain a changing set of variables and constraints, which must be represented as dynamic constraint networks.

The importance of dynamic constraint reasoning is well known. Many researchers have explored dynamic constraint problems and developed techniques to reason about them. There are two standard definitions for dynamic constraint problems. Mittal and Falkenhainer [10] define them as a set of variables and constraints, in which variable assignments determine whether parts of the problem are active or not. Dechter and Dechter's definition [3] is more general, as the sets of possible constraints and variables are not enumerated beforehand. However, the definition is limited to a predetermined domain for each variable. It is also worth noting that much of the work done in dynamic constraint reasoning has not considered dynamic problems which evolve by new variables being added.

In the most general sense, the problem of dynamic constraint reasoning can be viewed as providing constraint reasoning capabilities to a higher-level task that involves an underlying dynamic constraint problem. The high-level task in question can be a complex planning problem, a configuration problem, a design synthesis problem, a diagnosis problem, or any of a number of other important autonomous reasoning applications. The responsibility of the dynamic constraint reasoning framework is to represent the network, support modifications to the network, and to provide information about the state of the network. The general architecture is shown in Figure 1.

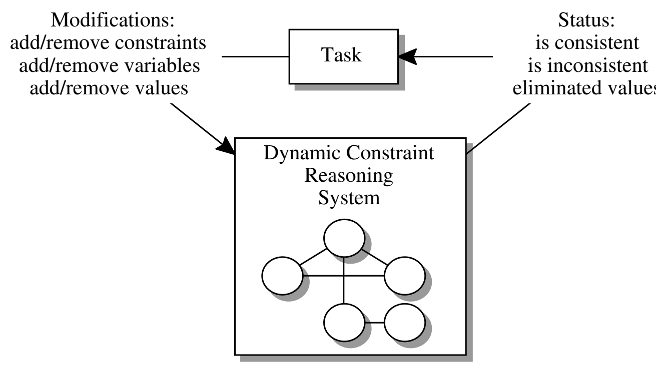

Figure 1. The architecture for dynamic constraint reasoning in support of a higher-level task involving a dynamically changing constraint network.

Recent developments in autonomous reasoning for complex real-world problems have pointed out significant shortcomings in current dynamic constraint reasoning techniques. This has been crystallized in the development of the Remote Agent planning technology, which is part of an effort to build autonomous control systems for remote spacecraft. The original Remote Agent planner was tested onboard the Deep Space One spacecraft in a landmark experiment that took place in May 1999. Since then, the Remote Agent planning technology has been under continuous development, both to formalize and clarify the constraint-based planning approach, and to build a reusable framework for future applications.

The requirements for such real-world applications have been well beyond the capabilities of existing dynamic constraint reasoning systems. The complexity of the applications, combined with the desire

[1] Research Institute for Advanced Computer Science, NASA Ames Research Center, Mailstop 269-2, Moffett Field, CA 94035, email: jonsson@ptolemy.arc.nasa.gov

[2] QSS Group, Inc., NASA Ames Research Center, Mailstop 269-3, Moffett Field, CA 94035, email: frank@ptolemy.arc.nasa.gov

[3] Static constraint reasoning has been used successfully to solve bounded planning problems, but those are of lower complexity than general planning problems.

for a domain-independent system, requires that the constraint reasoning system be able to represent and effectively utilize arbitrary constraints. This includes specialized reasoning methods for domain-specific elements, such as solving arithmetic equations. In addition, real-valued variables must be represented and reasoned about.

In this paper, we outline a general framework for dynamic constraint reasoning, with many capabilities that are missing from traditional dynamic constraint approaches. Among those are:

- Arbitrary constraints – there is no limit on the types of constraints that can be handled
- Dynamic variables – the set of variables need not be enumerated beforehand
- Dynamic domains – the set of values for certain variables, such as those representing dynamic entities, need not be enumerated beforehand
- Real-valued variables – constraints may involve mixtures of discrete and continuous variables
- Hybrid reasoning – different reasoning techniques can be utilized within the same constraint network

The resulting framework has been implemented and incorporated into the next generation Remote Agent planning system [8]. 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 Remote Agent planner in the next section, as an example of a complex real-world autonomous reasoning system that requires all the capabilities of our framework. We then go on to give a quick overview of the key concepts in dynamic constraint reasoning. Next, we turn our attention to the concept of procedural reasoning, which is the foundation on which our framework is built. In the following section, we present the overall framework and briefly mention some of its properties and capabilities. We conclude by reviewing our contributions and how this work will progress in the future.

2 The Remote Agent Planning Approach

To motivate the need for the capabilities provided by our dynamic constraint reasoning framework, we look at the Remote Agent (RA) planning paradigm [9].

In the RA planning approach, the world is described with *state variables* (also called *timelines*), whose values change over time. The values for a given state variable are the possible actions and states for that state variable. These values are described in terms of predicates, which provide a uniform representation of both complex actions and parameterized states. For an example, consider a state variable that describes the attitude of the spacecraft. The possible values for this state variable are `turn(x,y)`, to represent that the spacecraft is turning from coordinate `x` to coordinate `y`, and `pointing(x)` to represent that the spacecraft maintains its orientation towards coordinate `x`.

The states and actions have temporal extents. To represent this, *intervals* are used to describe a state variable maintaining a value for some duration. Each interval is described using a set of CSP variables. The temporal aspects of an interval are described using variables that represent the start time, the end time, and the duration of the interval. The remaining aspects, the parameters of the predicate, are also described by a set of CSP variables. Based on this notion, the evolution of a state variable is described by a sequence of intervals, connected by temporal constraints.

Planning axioms specify relations between intervals, enforcing preconditions, effects, enabling conditions, mutual exclusions, etc. These axioms give rise to constraints between the different CSP variables that describe the different intervals. The constraints can be arbitrarily complex, and may be domain-dependent, such as non-linear numerical bounds on available solar power. The use of exhaustive enumeration of tuples for such constraints, while theoretically useful, is too inefficient for real applications. In addition, applications of the RA planner, like many other real-world applications, often involve real-valued variables. Although many constraints over continuous variables have no closed solutions, the constraint reasoning component must nonetheless be able to handle such variables in a well-behaved and effective manner.

The planning axioms also give rise to new intervals, to achieve preconditions, describe effects, and so on. Each interval must be placed on an appropriate state variable, as part of the planning process. Constrained variables are used to describe the possible placement of subgoal intervals. Note that the domain of those variables must be extendable, as the addition of other intervals on the same state variable may change the set of possible locations[4].

The variables and the constraints used in this planning approach give rise to a constraint network for each candidate plan. In order for the candidate plan to be valid, the underlying constraint problem must have a valid solution. As planning decisions are made, the constraint network changes; adding intervals leads to new variables, values and constraints, while removing intervals results in variables, values and constraints being removed.

Supporting the dynamic constraint reasoning for such complex real-world applications requires extensive capabilities. The higher complexity of planning makes it infeasible to specify the set of variables that may be encountered. The generality of the approach requires the ability to handle arbitrary constraints, including complex relations that cannot be specified by enumerating the allowed or disallowed combinations. Furthermore, the network may involve real-valued variables and specialized reasoning methods to handle those. The constraint network is also highly dynamic, as there are no bounds on the modifications made by the higher-level planning task. Not only do the changes depend on the planning domain, but also on the search technique employed in the high-level reasoning task. Finally, the constraint reasoning must be correct and efficient.

3 Dynamic Constraint Reasoning

A constraint satisfaction problem can be thought of as 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,

Definition 1 (CSP) *A* constraint satisfaction problem *(CSP) is a triple $C = (X, D, K)$, where:*

1. *$X = \{x_1, \ldots, x_n\}$ is a set of variables*
2. *$D = \{D_{x_1}, \ldots, D_{x_n}\}$ a set of variable domains, one for each variable*
3. *K is a set of constraints, each of which is a pair $k = \langle Y, R \rangle$, where $Y = x_{i_1}, \ldots, x_{i_k}$ is the* scope *of the constraint, and R is a relation over the domains of the scope, i.e, $R \subset D_{x_{i_1}} \times \cdots \times D_{x_{i_k}}$*

A solution to a constraint satisfaction problem is a complete set of assignments of values to variables, such that all constraints are satisfied:

[4] See [5] for a recently developed alternative to using dynamic domains.

Definition 2 (Solution) *A* valid solution *to a constraint satisfaction problem $C = (X, D, K)$, where $X = \{x_1, \ldots, x_n\}$, 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 $(Y, R) \in K$ 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*. If every combination of variable assignments is a solution, the constraint problem is *solved*.

In many applications with underlying constraint problems, the constraint network changes as higher-level 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 a sequence of individual static problems. Current formalizations of dynamic constraint networks are too weak to describe the changes that may occur in a complex domain, e.g., dynamic domains are not handled. We therefore provide a definition of dynamic constraint satisfaction that is more general than previous definitions.

Definition 3 (Dynamic CSP) *Let $C = (X, D, K)$ be a constraint satisfaction problem. Any problem of the form $C' = (X', D', K')$ such that $X' \supseteq X$ (i.e. there are more variables), $D'_x \subseteq D_x$ for each $x \in X$ (i.e. there are fewer legal values for variables) and $K' \subseteq K$, (i.e. there are fewer legal combinations for variables in a constraint) is a* restriction *of C. Any problem of the form $C' = (X', D', K'')$ such that $X' \subseteq X$ (i.e. there are fewer variables), $D'_x \supseteq D_x$ for each $x \in X$ (i.e. there are more values for variables) and $K' \supseteq K$ (i.e. there are more legal combinations for variables in a constraint), is a* relaxation *of C. A* Dynamic Constraint Satisfaction Problem *or DCSP is a sequence of constraint satisfaction problems $C_0, C_1, \ldots$, such that each problem C_i is a* restriction *or a* relaxation *of C_{i-1}.*

This definition allows us both to add and delete variables, and to add and delete domain values, in addition to modifying the set of constraints.

We are mainly interested in the consistency of each constraint problem instance. The reason is that an inconsistent underlying constraint network invariably indicates that the higher-level solution candidate is invalid.

4 Procedural Reasoning

Many applications of dynamic constraint reasoning are designed and implemented for specific domains with predetermined types of variables, domains and constraints. In contrast, general-purpose model-based reasoning system can give rise to arbitrary domains with arbitrary constraints. To support such systems, and to retain generality, our dynamic constraint reasoning mechanism 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 many autonomous control applications. To achieve this, we utilize the notion of *procedural reasoning*, which has recently been formalized in the context of constraint satisfaction [7]. Procedures were initially developed to make constraint reasoning more effective, but 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.

The most general notion of a procedure encompasses a wide range of constraint reasoning techniques, from simple propagation to complete search methods:

Definition 4 (Procedure) *A* procedure *p is a function that maps a CSP $C = (X, D, K)$ to another CSP $C' = (X, D', K')$ that has the same variables as C, such that:*

1. *$D'_{x_i} \subseteq D_{x_i}$ for each x_i*
2. *For every $k = (Y, R) \in K$ there exists a constraint $k' = (Y, R') \in K'$, such that k and k' have the same scope, and $R' \subseteq R$.*

This definition permits a procedure to eliminate values from variable domains, restrict existing constraints, and add new constraints.

Since procedures are so general, we need to distinguish between procedures, based on their effects on the set of solutions. At a first glance, one might expect that a procedure should preserve all solutions to a constraint network. However, there are many useful procedures, such as symmetry breaking, that do not satisfy this strong criterion. Such procedures may eliminate some solutions, but are guaranteed to not eliminate all solutions. Formally:

Definition 5 (Correctness) *A procedure p is* correct *if the set of solutions to $p(C)$ is the same as the set of solutions to C. A procedure p is* weakly correct *if $p(C)$ has a solution whenever C has a solution.*

The general notion of a procedure permits the addition of arbitrary numbers of new constraints to the problem. For many applications, such problem growth cannot be handled effectively. Therefore, a restricted class of procedures is defined, consisting of procedures that cannot add any new constraints. As a result, such procedures are limited to eliminating values from variable domains:

Definition 6 (Elimination Procedure) *An* elimination procedure *p is a procedure that preserves the set of constraints. In other words, $p(X, D, K) = (X, D', K)$.*

In the currently implemented framework, we have limited ourselves to elimination procedures, because of the cost associated with added constraints, and the unavoidable utility problem that follows.

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

Certain classes of procedures and search engines have useful theoretical properties, most notably regarding correctness, systematicity and completeness. Jónsson [7] proved that, for a large class of complete and systematic algorithms, the use of correct procedures will not affect completeness or systematicity, and 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 [6], satisfy these conditions.

In many constraint reasoning problems, it is only necessary to assign values to a subset of the variables to determine consistency. For example, in satisfiability algorithms like Davis-Putnam [2], as soon as all clauses are satisfied, there is no need to instantiate remaining variables. This notion is formalized in the concept of a decision set:

Definition 7 (Decision Set) *Let p be a weakly correct procedure and let $C = (X, D, K)$ be a CSP. A set of variables $Y \subset X$ is a* decision set *for C with respect to p, if, for any partial assignment A to all the variables in Y, applying p to C_A results either in a CSP that is solved or a CSP with an empty variable domain.*

To solve a problem, it is sufficient to assign values to variables in the decision set, as the procedure can be used to determine if the rest of the problem is solvable. This 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 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.

5 Framework for Dynamic Procedural Reasoning

The concepts we have introduced, procedures and decision sets, are the foundation of our general dynamic constraint reasoning framework. Procedures give us a uniform and efficient method to represent and apply constraints, while decision sets provide a general method to support continuously valued variables.

Our implemented framework has three main purposes. One is to represent the underlying constraint network, i.e., the variables, the sets of possible values, and the constraints. The second is to support network modifications, which are driven by a higher-level task. The third responsibility is to determine or approximate the state of the network, in particular regarding consistency or inconsistency.

In terms of modifying the network, the framework permits almost arbitrary changes to be made:

- Add new variables – with discrete or real domains
- Remove existing variables
- Add new constraints – declarative or procedural
- Remove existing constraints
- Eliminate values from variable domains
- Restore previously eliminated values
- Add new values to variable domains

Our framework supports 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. Currently, those are:

- Equality and inequality constraints
- Distance constraints for temporal variables[5]

All non-declarative constraints are enforced by elimination procedures. This approach allows us to incorporate arbitrary constraints into the constraint network, without any changes to the constraint reasoning engine. As was pointed out above, any constraint can be formulated as an elimination procedure. To specify a new constraint, all that needs to be done is to implement an elimination procedure that enforces the constraint and then add the procedure directly to the constraint network. The expressiveness of the framework is therefore not limited to a predetermined set of constraints, which makes it truly applicable to different problem domains.

Our framework also supports real-valued variables. Unfortunately, sound and complete reasoning with arbitrary procedural constraints over continuous variables is not possible. We therefore prohibit real variables as decision variables. This means that the higher-level task does not have to (but is allowed to) directly modify real-valued domains in order to build a solved network. Instead, once a valid solution is found to the set of decision variables, the procedures will determine the valid values for non-decision variables, resulting in either a solved network or an inconsistency. Note that this does not preclude real-valued variables from participating in constraints, and thus having an effect on the solvability of the problem.

The third main responsibility of our constraint reasoning framework is to handle queries about problem instances. The information provided in response to such queries includes:

- Whether the network is consistent – a solution exists
- Whether the network is inconsistent – no solution exists
- Which values can be eliminated – values proven not to be part of any solutions

Due to the inherent intractability of determining these questions exactly, the framework allows for approximate (but sound) answers[6]. Examples of sound approximations include only indicating that the instance has not been proven to be inconsistent, and providing only a subset of the values that can provably be eliminated from a domain. This trades off 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 [1], and providing the approximate answers based on the results.

The simplest approximation is based on a variation of a simple algorithm for maintaining generalized arc-consistency. Given a change in the constraint network, the algorithm determines a set of variables W, whose set of consistent values may have been affected. If the change is a combination of restrictions, finding W is straightforward; enumerate new variables, variables appearing in new constraints and variables with reduced domains. If the change includes a relaxation, i.e., an expanded domain or a relaxed constraint, it is more difficult to specify a suitable W. Of course, the full set of variables is a valid candidate, but it is often unnecessarily large. To reduce this set, dependency information, based on recording which variables affect other variables through constraints, can be used. For procedural constraints, a sound assumption is that each variable affects every other variable in the scope. However, the dependency information can be strengthened by extending procedural constraints to provide dependency information. (See [7] for an example of extending procedures to provide a specific type of dependency information.)

Given a set of variables W that includes all variables affected by the change in the network, the algorithm works as follows:

1. if the network has been relaxed, reinstate all eliminated values for all variables in W
2. let A be the set of all constraints with scopes that overlap W
3. while A is not empty, do:
   a) select and remove a constraint c from A
   b) execute c to eliminate values from domains
   c) if a domain becomes empty, return "inconsistent"
   d) for any decision variable v whose domain is modified, add to A all constraints with v in their scope
4. return "not proven inconsistent"

[5] The temporal constraints form a simple temporal network which can be handled using efficient reasoning techniques, as described in [4] and [12].

[6] Methods to determine consistency can easily be incorporated into this framework, but such methods are rarely practical when supporting higher-level problem solving.

The result returned by this algorithm is provably correct, given that each procedure is at least weakly correct. Furthermore, if each constraint is correct, then no values participating in solutions have been eliminated.

It should be noted that the algorithm will not achieve general arc-consistency in all cases. The reason is that the procedural constraints are not required to achieve arc-consistency when eliminating values from the domains. However, it is straightforward to show that if each procedural constraint enforces general arc-consistency internally, the overall network will be arc-consistent.

The computational complexity of this algorithm is obviously tied to the complexity of the procedural constraints in the network. Let n be the number of variables in the network, let d be the size of the largest discrete domain, let r be the arity of the largest constraint and let e be the number of constraints. Then, if the worst-case time complexity of executing a single procedure is $O(f(r,d))$, the overall algorithm runs in time $O(d \cdot n \cdot e \cdot f(r,d))$. Replacing the procedure with a table-lookup that takes $O(d^r)$ in the worst case, we find that the complexity is not as good as other versions of generalized arc-consistency. This is not surprising, as no assumptions are made about the constraint structures or efficiency. However, in almost all cases, the worst-case complexity of the procedures, $f(r,d)$, will be significantly less than d^r. This is because the procedure can directly calculate which values can be eliminated, rather than search for possible support for them. Furthermore, procedures can take advantage of characteristics like the constraint being functional, symmetric, row-convex, etc. Finally, many of the speedup techniques in arc consistency, such as maintaining support lists [11], can be incorporated into the procedures.

The overall functionality of the consistency achievement algorithm is the same as that of a single procedure; namely, to eliminate values and to identify inconsistencies. Consequently, the algorithm can be viewed as a method for composing multiple procedures into a single procedure. In the above algorithm, the composition is done by repeated applications of the constraints until local quiescence. Other combination methods may also be used, and as long as they are limited to methods that only eliminate values, such as inverse local consistency, the worst-case computational complexity will remain bounded by the complexity of the method, multiplied by the complexity of executing the constraint procedures.

Our dynamic constraint reasoning framework has been implemented and integrated into the new Remote Agent planner, providing the new planner with a well-founded and effective dynamic constraint reasoning mechanism. The generality of the framework has been demonstrated by duplicating the constraint reasoning done during the Remote Agent Experiment, and by applying it to other complex domains, such as scheduling observations and downloads for spacecraft with limited data storage capacity.

6 Conclusions and Future Work

The goal of this work is to develop a general-purpose framework for performing dynamic constraint reasoning in real-world applications. Such applications give rise to a number of issues that have traditionally not been handled in dynamic constraint reasoning, such as special-purpose, domain-dependent constraints that are too complex to be specified in tabular form. Many complex decision problems also require that real-valued variables be handled, and although such variables rarely represent primary choices in the automated reasoning, they must still be handled efficiently and effectively. Finally, real-world applications demand more dynamicity than traditional definitions permit. For example, the variables involved in a planning problem cannot be specified beforehand, as the number of actions is only bounded by an exponential function. The value domains cannot be predetermined either, as variables are often used to represent dynamic entities, such as available actions or locations in action sequences.

In this paper, we have outlined an implemented framework for dynamic constraint reasoning, based on the notion of procedures. Procedures allow us to represent constraints in a uniform manner, without requiring inefficient declarative specifications. They also enable special-purpose reasoning techniques, such as solving linear equations, to be specified as part of the constraint network. The notion of decision sets allows us to include real-valued variables, without losing soundness or completeness. The result is a general, well-defined framework for dynamic constraint reasoning that can be applied in a number of different real-world problem domains.

Given the generality of the framework, much of our future work is in applying it within different applications. Among the possible candidates are other complex planning and scheduling systems, configuration systems, mixed-initiative problem solving systems and intelligent design systems. There is also more work to be done on the theoretical side. For example, there is the question of which consistency-maintenance techniques can be used effectively with procedural constraints. It is clear that overall consistency and inverse neighborhood-consistency can be enforced by using the appropriate search techniques in lieu of enforcing arc-consistency.

REFERENCES

[1] Christian Bessière and Jean-Charles Régin, 'Arc consistency for general constraint networks: Preliminary results', in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pp. 398–404, (1997).

[2] Martin Davis and Hilary Putman, 'A computing procedure for quantification theory', *Journal of the ACM*, **7**, 201–215, (1960).

[3] Rina Dechter and Avi Dechter, 'Belief maintenance in dynamic constraint networks', *Proceedings of the Seventh National Conference on Artificial Intelligence*, 37–42, (1988).

[4] Rina Dechter, Itay Meiri, and Judea Pearl, 'Temporal constraint networks', *Artificial Intelligence*, **49**, 61–95, (1991).

[5] Jeremy D. Frank, Ari K. Jónsson, and Paul H. Morris, 'On reformulating planning as dynamic constraint satisfaction', in *Proceedings of Symposium on Abstraction, Reformulation and Approximation*, (2000).

[6] Matthew L. Ginsberg, 'Dynamic backtracking', *Journal of Artificial Intelligence Research*, **1**, 25–46, (1993).

[7] Ari K. Jónsson, *Procedural Reasoning in Constraint Satisfaction*, Ph.D. dissertation, Stanford University, Stanford, CA, 1997.

[8] 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)*, pp. 363–370, (1999).

[9] Ari K. Jónsson, Paul H. Morris, Nicola Muscettola, Kanna Rajan, and Ben Smith, 'Planning in interplanetary space: Theory and practice', in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pp. 177–186, (2000).

[10] Sanjay Mittal and Brian Falkenhainer, 'Dynamic constraint satisfaction problems', in *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 25–32, (1990).

[11] Roger Mohr and Thomas C. Henderson, 'Arc and path consistency revisited', *"Artificial Intelligence"*, **28**, (1986).

[12] Ioannis Tsamardinos, Nicola Muscettola, and Paul Morris, 'Fast transformation of temporal plans for efficient execution', in *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 254–261, (1998).