On Reformulating Planning As Dynamic Constraint Satisfaction (Extended Abstract)

Jeremy Frank* Ari K. Jónsson** Paul Morris **
NASA Ames Research Center
Mail Stop 269-1
Moffett Field, CA 94035
{frank,jonsson,pmorris}@ptolemy.arc.nasa.gov

Abstract. In recent years, researchers have reformulated STRIPS planning problems as SAT problems or CSPs. In this paper, we discuss the Constraint-Based Interval Planning (CBIP) paradigm, which can represent planning problems incorporating interval time and resources. We describe how to reformulate mutual exclusion constraints for a CBIP-based system, the Extendible Uniform Remote Operations Planner Architecture (EUROPA). We show that reformulations involving dynamic variable domains restrict the algorithms which can be used to solve the resulting DCSP. We present an alternative formulation which does not employ dynamic domains, and describe the relative merits of the different reformulations.

1 Introduction

In recent years, researchers have investigated the reformulation of planning problems as constraint satisfaction problems (CSPs) in an attempt to use powerful algorithms for constraint satisfaction to find plans more efficiently. Typically, each CSP represents the problem of finding a plan with a fixed number of steps. A solution to the CSP can be mapped back to a plan; if no solution exists, the number of steps permitted in the plan is increased and a new CSP is generated. SATPlan [SK96] mapped planning problems in the STRIPS formalism into Boolean Satisfiability (SAT) problems. Early versions required hand-crafted translation of each planning domain in order to achieve good problem solving performance; later, automated translation of arbitrary STRIPS domains into SAT problems achieved good performance as well [ME97]. Graphplan [BF97] works on STRIPS domains by creating a *plan graph* which represents the set of propositions which can be achieved after a number of steps along with mutual exclusion relationships between propositions and actions. This structure is then searched for a plan which achieves the goals from the initial condition. While the original algorithm performed backward search, the plan graph can also be transformed into a CSP which can be solved by any CSP algorithm [DK00].

* QSS Group, Inc
** Research Institute for Advanced Computer Science

A second growing trend in planning is the extension of planning systems to reason about both time and resources. STRIPS is simply not expressive enough to represent more realistic planning problems. This demand for increased sophistication has led to the need for more powerful techniques to reason about time and resources during planning. The scheduling community has used constraint satisfaction techniques to perform this sort of reasoning. Coupled with the successes achieved by reformulating STRIPS problems, this provides incentives to consider reformulating more complex planning domains as CSPs.

There have been several efforts to create planners which reason about time and resources, and many such planners employ an underlying constraint reasoning system to manage complex constraints during planning. These planners use interval representations of time and often use constraint systems to manage temporal and resource constraints; [SFJ00] refers to systems like these as Constraint-Based Interval Planners (CBIPs). ZENO [Pen93] and Descartes [Jos96] are important examples of such planners; unfortunately, space limitations prohibit us from doing more than mentioning these efforts. HSTS [Mus94] employs an interval representation of time and permits arbitrary constraints on the parameters of actions. Temporal constraints and parameter constraints are reformulated as a DCSP. At each stage in planning, the DCSP is made arc consistent, and inconsistencies result in pruning. HSTS also adds the notions of attributes and timelines. An attribute is a subsystem or component of a planning domain; timelines represent sequences of actions or states on attributes. Attributes permit more intuitive modeling of planning domains, and enable the enforcement of mutual exclusion. Finally, HSTS employs a unique, uniform representation of states and actions. The Remote Agent Planner (RAP) [JMM$^+$00] employs the above mechanisms as part of the control system for the Deep Space One spacecraft in May of 1999. The Extendible Uniform Remote Operations Planner Architecture (EUROPA) is the successor of RAP. An important goal of EUROPA is to support a wide variety of search algorithms. EUROPA maps the entire planning problem into a DCSP, providing explicit variables for subgoal decisions as well as conditional subgoaling. In addition, due to the size and complexity of non-binary constraints used in space applications, EUROPA uses *procedural constraints* [Jón97, JF00] to represent the underlying DCSP.

Much of the reformulation of a CBIP-based planning problem as a DCSP is straightforward. The temporal components in the plan can often be represented as a Simple Temporal Network [DMP91], and complex constraints such as resource constraints can be implemented as procedural constraints [Jón97]. Disjunctions can be modeled directly by variables whose domains represent the possible choices, as is done in EUROPA. However, the addition of mutual exclusion complicates the task of reformulating CBIP domains. The obvious way of enforcing the mutual exclusion constraints leads to a DCSP representation using dynamic variable domains. This representation makes reasoning about no-goods quite difficult; since many important enhancements to search algorithms depend on no-good reasoning, this is a serious drawback. In this paper we first describe the CBIP paradigm and EUROPA, then describe how introducing mutual exclusion leads to these complications. We then show how to represent mutual exclusion constraints as a DCSP without dynamic domains. Finally, we discuss the impact of this representation on algorithms to solve the resulting DCSP.

2 Constraint-Based Interval Planning

The Constraint-Based Interval Planning (CBIP) framework is based on an interval representation of time. A *predicate* is a uniform representation of actions and states, and an *interval* is the period during which a predicate holds. A *token* is used to represent a predicate which holds during an interval. Each token is defined by the start, end and duration of the interval it occurs, as well as other parameters which further elaborate on the predicate. For instance, a `thrust` predicate may have a parameter describing the thrust level, which can be either `low, medium or high`. The planning domain is described by *planning schemata* which specify, for each token, other tokens that must exist (e.g. pre and post conditions), and how the tokens are related to each other. Figure 1 shows an example of a planning schema. Schemata can specify conditional effects and disjunctions of required tokens. For instance in Figure 1, a thrust interval can be met by a short warmup period if the engine is already warm, or a longer one if not. Variables representing the disjunctions are parameters of tokens, and thus are DCSP variables. This is shown in Figure 1, as the value of the `?temp` variable indicates the duration of the `warmup` token which precedes the `thrust` token. Planning schemata can also include constraints on the parameters of the token. As shown in Figure 1, the thrust interval has a constraint relating the thrust level, available fuel, and the duration.

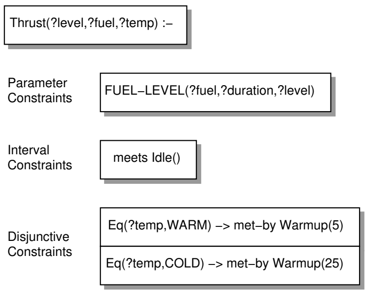

Fig. 1. The planning schema for a thrust interval. This schema consists of four components: the master token of the schema, constraints on the parameters of the schema, a description of other tokens which must exist when the master token is in the plan, and a disjunction of tokens which may exist when the master token is in the plan.

EUROPA is a CBIP planning paradigm which continuously reformulates the planning problem as a DCSP problem. This is done by mapping each partial plan to a CSP. The temporal constraints form a Simple Temporal Network, which can be efficiently solved [DMP91], while the rest of the constraints form a general, non-binary CSP represented by procedural constraints [JF00]. Figure 2 shows a small partial plan and its induced CSP. Assignments of variables in the CSP correspond either to the adding of new plan steps, or the assignment of parameters of plan steps. As steps are added to or removed from the plan, the CSP is updated to reflect the current partial plan. For example, in Figure 1, adding the `thrust` step to the plan requires adding several new variables and constraints to the CSP. At any time, if the CSP is inconsistent, then the partial plan it represents is invalid; if a solution is found to the CSP, then that solution can be mapped back to a plan which solves the problem. The advantage of such a representation is that any algorithm which solves DCSPs can be used to solve the planning problem.

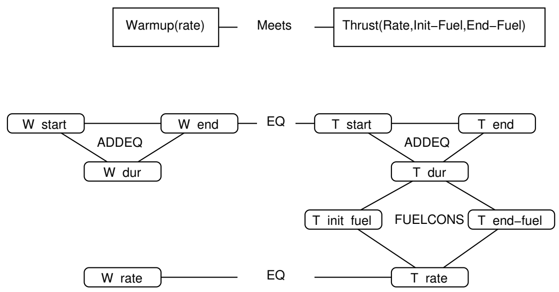

Fig. 2. A partial plan and its DCSP representation. The partial plan consists of 2 tokens, shown at the top of the figure. The DCSP variables are in rounded boxes. Edges between DCSP variables are labeled with the constraints on those variables.

3 Timelines in EUROPA: Square Tokens and Round Slots

EUROPA represents attributes of planning domains using *timelines*. Timelines are ordered sequences of token equivalence classes, which represent how an attribute changes during the course of a plan. This adds powerful constraints to the planning problem, which make it possible to eliminate a large number of candidate solutions. Also, the specification of planning domains is more natural than in languages such as STRIPS. However, the planning domain must now specify which tokens can appear on which timeline; this requires a more sophisticated domain model. The planner framework must also contain a mechanism for enforcing mutual exclusions.

Adding an action to a plan requires inserting a token onto a timeline. A *slot* is a legal place on a timeline where a token can be inserted. Tokens can only be inserted into single slots; they can't span multiple slots. Each token equivalence class defines a *full* slot, and there is an *empty* slot between each pair of sequential token equivalence classes. When a token is inserted into an empty slot, new empty slots are created before and after the token. However, when a token is inserted onto a full slot, no new slots are created. Instead, the start timepoint and end timepoint of the new token are equated with the timepoints of the tokens defining the slot, and all the parameters are equated to the parameters of the tokens on the slot.

Timelines enforce mutual exclusion among tokens with different predicates. This models the notion of an attribute maintaining only one state at a time, such as a unit resource which can only be used by a single task at once in a scheduling problem. Timelines enforce a partial order among tokens; either a token is strictly before or strictly after another token, or it occupies exactly the same interval (or slot) as another token, which is another way of saying that the two tokens specify the same action or state. This ensures that incompatible actions are not permitted to overlap on the same timeline.

4 Representing Mutual Exclusion in EUROPA

The description of timelines leads to a natural representation of mutual exclusion constraints in EUROPA. Each token insertion decision is represented by a variable. The domain of this variable is the set of slots on a timeline. Notice, however, that this domain is *dynamic*, as the set of available slots changes as new tokens are inserted onto timelines. If search were guaranteed to proceed chronologically, the search algorithm could simply store the previous domains for the slots. However, EUROPA is designed to support many search algorithms, including non-chronological algorithms. This means that timelines can change in arbitrarily complex ways as the search for a plan proceeds. Identifying an arbitrary slot as one which occurred in a previous plan state would require saving all intermediate plan states, as well as performing expensive matching operations. This means that new labels for slots must be generated as timelines evolve.

While tokens can nominally be inserted into any slot on a timeline, in practice there are usually very few options which do not immediately lead to a constraint violation. For instance, some slots may be occupied by tokens with incompatible predicates, while other slots may simply be too small (such as slots of zero duration between adjacent tokens on a timeline). Lookahead mechanisms can rapidly reduce the set of candidate slots. There are a number of possible ways to implement this lookahead; checking predicates is inexpensive, while checking temporal constraints and parameter constraints is more expensive.

Figure 3 shows an example of how lookahead can be done. In this example, the thrust token has a duration of between 4 and 6. Simply by checking the predicate names, a lookahead mechanism can eliminate slots 1 and 5. If the mechanism checks the legal start and end times for the token, slot 4 is eliminated, because the token must end before slot 4 begins. If the mechanism checks the duration of slot 3, it would find it was too short, having a maximum duration of 3. This leaves slots 2 as the only candidate.

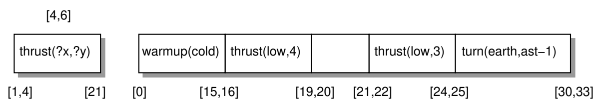

Fig. 3. Checking for suitable slots. The free `thrust` token at the left of the figure has a duration of between 4 and 6, and its start and end times are also given. Simple lookahead can eliminate all candidate slots except slot 2. Note that slots of zero duration between adjacent tokens are not represented in this figure.

This representation has some subtle but important ramifications for sophisticated CSP algorithms. Consider, for example, powerful no-good learning techniques employed by algorithms such as Dynamic Backtracking [Gin93], RelSat [BM96], and Tabu search [Glo89]. A no-good is simply a combination of variable assignments which cannot be part of a solution. No-goods containing values from dynamic domains are, unfortunately, "no good" when the value changes during search. To see why, consider a no-good containing a token insertion onto an empty slot. The value representing the empty slot will be eliminated from the domain of the token insertion variable and replaced with new values representing the new slots. Even should the token be removed later, the domain of this variable will be updated with new values, because of the expense of inferring that the labels should be identical. Since the domain can change many times as a succession of different tokens are inserted into the empty slot, no-goods using the empty slot value may not be usable, because they will not match the current context if the value in the no-good has been replaced.

5 The Ordering Decision Representation

In this section we propose a mutual exclusion representation which uses boolean variables to represent decisions about the order of tokens on a timeline. Recall that timelines are an ordered list of token equivalence classes which define the slots. In effect, the slots are a consequence of committing to one of the possible orderings of the tokens. As we saw above, these slots are mutable, and thus representations which depend explicitly on the identity of the slots will suffer from the problems with dynamic domains. A representation based on ordering decisions among tokens on the same timeline does not have this problem. As new tokens are added, new variables are added, but their domains are not dynamic.

We now describe the new representation in detail. When a new token A is introduced, we create 3 boolean variables describing the relationship between this token and each other token B: $Bef(A, B)$, $Aft(A, B)$, and $Eq(A, B)$. We must also create a number of *conditional constraints* which relate a boolean ordering variable and timepoint variables for A and B. These constraints permit information about the boolean ordering variables to affect the possible values of the timepoints, and vice-versa. For instance, if $(Bef(A, B) = T)$, the conditional constraint would enforce $(e_A \leq s_B)$. Similarly, if $(e_A > s_B)$, the conditional constraint would enforce $(Bef(A, B) \neq T)$. To see how the representation using conditional constraints works, let s_A, s_B be the start timepoints of tokens A, B respectively, and e_A, e_B be the end timepoints of tokens A, B respectively. To enforce the total ordering of A and B, we use the following conditional constraints:

$$(Bef(A,B) = T) \Rightarrow (e_A \leq s_B)$$

$$(Aft(A,B) = T) \Rightarrow (e_B \leq s_A)$$

The case for $Eq(A,B) = T$ is a bit more complex. Recall that tokens have parameter variables as well as temporal variables; let a_i be the i^{th} parameter of A and b_i be the i^{th} parameter of B respectively. Then we have the following constraints:

$$(Eq(A,B) = T) \Rightarrow (s_A = s_B)$$

$$(Eq(A,B) = T) \Rightarrow (e_A = e_B)$$

$$(Eq(A,B) = T) \Rightarrow \forall i(a_i = b_i)$$

Recall that we pose these constraints between every pair of tokens on the same timeline.

We can exploit the fact that only one of $Bef(A,B), Aft(A,B)$ and $Eq(A,B)$ can be true for any pair of tokens A and B, and post an additional XOR constraint between these three variables. Recall that some tokens have incompatible predicates. Such tokens must be totally ordered on a timeline; for these pairs, we post the unary constraint $Eq(A,B) = F$. Figure 4 shows the new representation.

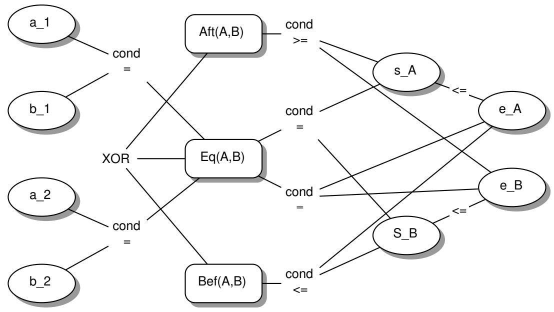

Fig. 4. Order variables and constraints for two tokens. Variables are represented by ovals, constraints are represented by labeled hyper-arcs.

If we recall the lookahead mechanism described in the previous section, we see that most of the lookahead operations are now subsumed by arc consistency. For example, incompatible predicates are handled by the unary constraints posted on the $Eq(A,B)$ variables. If a slot is too early or too late, then the conditional constraints will propagate that information to the boolean variables. The conditional constraints on the $Eq(A,B)$ variable will also result in propagation to eliminate full slot insertions which would cause constraint violations. The only lookahead check which is not immediately handled by propagation is the check on the duration of empty slots. The reason is that there are no constraints in the new formulation which mimic the duration constraints on slots. While these constraints could be posted, this would require inferring the location of the empty slots from the current order of tokens, which might be costly.

There are other ways to use variables and constraints to represent the mutual exclusion relationship, for instance using fewer variables. However, these representations lead to less intuitive, higher arity constraints. The representation we have chosen to discuss here has two advantages: it is relatively simple to explain, and the conditional constraints are general procedural constraints which fit well with the procedural constraint framework used in EUROPA.

For timelines with N tokens, there are $\frac{N(N-1)}{2}$ pairs of tokens, and each pair can be ordered 3 different ways. As such, the induced search space is $3^{\frac{N(N-1)}{2}}$. However, most of these possibilities are invalid, and can be eliminated after little search. For instance, if token A occurs before B and B occurs before C, then attempting to order A after C will quickly result in a temporal constraint violation. We can add optional constraints among the logical variables representing the ordering decisions to enable propagation which makes this search unnecessary. Consider the logical variables for tokens A, B and C. There are 13 possible arrangements of the tokens; either they are totally ordered (6 possibilities), 2 are equal and one comes either before or afterwards (6 possibilities), or all 3 are equated. We can post constraints like $Aft(A,B) \wedge Aft(B,C) \Rightarrow Aft(A,C)$ to enforce the conditions on total ordering of the tokens, $Eq(A,B)a \wedge Aft(B,C) \Rightarrow Aft(A,C)$ to enforce conditions on partial ordering of the tokens, and $Eq(A,B) \wedge Eq(B,C) \Rightarrow Eq(A,C)$ to enforce the conditions on all three equal. There are 13 total constraints; each time a new token is created for a timeline, we must add $\frac{13N(N-1)}{2}$ logical constraints on the new logical variables.

6 Comparing Representation

The original representation requires only a single variable to represent a token insertion decision. However, the domain for this variable is dynamic, and as we have seen, a special lookahead mechanism is necessary to reduce the domain. A label maintenance mechanism is also needed to update the names of elements of the domain as timelines evolve. Finally, this representation makes no-good reasoning difficult, beacuse many no-goods discovered during search may use values which are eliminated from the domain during search. These no-goods may not be used to best effect during search; the effort to collect these no-goods and match them to the current state is wasted overhead.

To assess the ordering representation, consider a timeline with N tokens inserted on it. The ordering variable representation requires $\frac{3N(N-1)}{2}$ logical variables, N XOR constraints, and $2N(N-1)$ conditional constraints on the timepoints and the logical variables. In addition, for each pair of tokens with identical predicates and p parameters, there are p conditional constraints between the parameter variables and the boolean variables representing the decision that two tokens have been equated. If the optional logical constraints are added, the contribution is $\frac{13N(N-1)(N-2)}{6}$ logical constraints. The main advantage of the ordering variable representation is that the mutual exclusion can be represented without using dynamic domains, so there are no problems with using algorithms such as Dynamic Backtracking or Tabu search. The increased search space is offset by the observation that constraint propagation limits the options for the ordering variables, so we expect to do roughly the same amount of search in the new representation.

One disadvantage of the new representation is that heuristic enforcement is more complicated. Natural heuristics for token insertion decisions are value-orderings, based on properties of slots such as relative order on the timeline, and whether the slot is full or empty. Since slots are no longer values of token insertion variables, this approach will not work. Enforcing these heuristics now requires dynamically ordering the boolean variables. For instance, to enforce a heuristic like "insert the token on full slots first" would mean specifying that the priority of assigning the $Eq(A, B)$ variables is higher than the priority of assigning the other boolean variables. Enforcing the heuristic "insert tokens onto the earliest slots first" would require determining which boolean variables correspond to decisions for tokens appearing earlier in the timeline, and giving priority to these variables.

7 Discussion and Future Work

Representing mutual exclusion constraints is an important component of the EUROPA reformulation of planning as constraint satisfaction. However, mutual exclusion reasoning complicates the automatic reformulation of planning domains into DCSPs. We have discussed two representations which manage mutual exclusion reasoning, and discussed some of the tradeoffs between these representations. Explicitly representing slots is intuitive, but results in a DCSP representation with dynamic domains, which leads to problems in using powerful CSP techniques such as no-good reasoning. Leveraging the power of existing CSP algorithms is a promising approach to solving planning algorithms. Our work is aimed at providing a representation which makes powerful no-good reasoning approaches feasible. We have presented an alternative representation which avoids the pitfalls of dynamic slot domains, but is more complex both in terms of the constraint network and in the enforcement of heuristics. It is premature to conclude that one approach is strictly superior to another.

The slot representation is one of the only instances of DCSPs employing dynamic domains we are aware of in the literature. Most such work only discusses adding and removing constraints among the same set of variables. Our observations concerning the pitfalls of no-good reasoning with the dynamic domain representation may be a manifestation of a deeper problem with dynamic domains, especially when values in these domains change over time. This phenomenon should be investigated more closely, and should it prove to be a pervasive problem, it will become important to consider ways of representing these problems without employing dynamic domains.

We would like to thank the anonymous reviewers for their comments.

References

[BF97] A. Blum and M. Furst. "fast planning through planning graph analysis". *Artificial Intelligence*, 90:281 – 300, 1997.

[BM96] R. Bayardo and D. Miranker. A complexity analysis of space bounded learning algorithms for the constraint satisfaction problem. *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 298–304, 1996.

[DK00] M. B. Do and S. Khambhampati. Solving planning-graph by compiling it into csp. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–94, 1991.

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

[Glo89] F. Glover. Tabu search: Part i. *ORSA Journal on Computing*, 1989.

[JF00] A. Jónsson and J. Frank. A framework for dynamic constraint reasoning using procedural constraints. *Euopean Conference on Artificial Intelligence (to appear)*, 2000.

[JMM+00] 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 and Scheduling*, 2000.

[Jón97] A. Jónsson. *Procedural Reasoning in Constraint Satisfaction*. PhD thesis, Stanford University Computer Science Department, 1997.

[Jos96] D. Joslin. *Passive and Active Decision Postponement in Plan Generation*. PhD thesis, Carnegie Mellon University Computer Science Department, 1996.

[ME97] D. Weld M. Ernst, T. Millstein. 1169-1176. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

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

[Pen93] S. Penberthy. *Planning with Continuous Change*. PhD thesis, University of Washington Department of Computer Science and Engineering, 1993.

[SFJ00] D. Smith, J. Frank, and A. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):61–94, 2000.

[SK96] B. Selman and H. Kautz. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 1194–1201, 1996.

This article was processed using the LaTeX macro package with LLNCS style