Control Knowledge in Planning: Benefits and Tradeoffs

Yi-Cheng Huang
Department of Computer Science
Cornell University
ychuang@cs.cornell.edu

Bart Selman
Department of Computer Science
Cornell University
selman@cs.cornell.edu

Henry Kautz
Shannon Laboratory
AT&T Labs - Research
kautz@research.att.com

Abstract

Recent new planning paradigms, such as Graphplan and Satplan, have been shown to outperform more traditional domain-independent planners. An interesting aspect of these planners is that they do not incorporate domain specific control knowledge, but instead rely on efficient graph-based or propositional representations and advanced search techniques. An alternative approach has been proposed in the TLPlan system. TLPlan is an example of a powerful planner incorporating declarative control specified in temporal logic formulas. We show how these control rules can be parsed into Satplan. Our empirical results show up to an order of magnitude speed up. We also provide a detailed comparison with TLPlan, and show how the search strategies in TLPlan lead to efficient plans in terms of the number of actions but with little or no parallelism. The Satplan and Graphplan formalisms on the other hand do find highly parallel plans, but are less effective in sequential domains. Our results enhance our understanding of the various tradeoffs in planning technology, and extend earlier work on control knowledge in the Satplan framework by Ernst *et al.* (1997) and Kautz and Selman (1998).

Introduction

In recent years, there has been a burst of activity in the planning community with the introduction of a new generation of constraint and graph-based methods, such as graphplan and satplan (Blum and Furst 1995; Kautz and Selman 1996; Kambhampati 1997; Weld 1999). These planners are domain-independent and outperform more traditional planners on a range of benchmark problems. The surprising effectiveness of these planners represents a departure from the long held believe that the use of domain-specific planning control knowledge is unavoidable during plan search. Nevertheless, control knowledge has the potential to significantly increase the performance of the new planners. In fact the constraint-based framework behind graphplan and satplan allows one, at least in principle, to incorporate control knowledge in a purely declarative manner by encoding the control as additional constraints.

A recent example of the effectiveness of declarative control knowledge is the TLPlan system by Bacchus and Kabanza (1996; 1998). In the TLPlan system, control knowledge is represented by formulas in temporal logic. For example, the "next" operator from temporal logic allows one to specify what can and cannot happen at the next time step. The control knowledge is used to steer a forward chaining planner. One of the surprises of this system is that, despite the rather basic search method, with the right control knowledge, the system is highly efficient on a range of benchmark problems, often outperforming Graphplan and Blackbox (Blackbox is the latest version of the Satplan system (Kautz and Selman 1999); Bacchus and Kabanza 1998). Of course, in this comparison Graphplan and Blackbox ran without any control; in addition, developing the right control formulas for TLPlan is a non-trivial task.

As Bacchus and Kabanza point out, the forward chaining approach is a good match with the declarative control specification. At each node in the search tree, the control formula is evaluated to determine what new nodes are reachable from the current state. With good control knowledge, many nodes are pruned and the search is "pushed" towards the goal state. To give some intuition as to how this is achieved, note that the control rules can encode information about the difference between the current state and the goal state by using a predicate that states, for example, "package currently not at goal location."

One interesting research question is whether the same level of control can be effectively incorporated into the Graphplan or Blackbox style planner. This is a non-trivial question because TLPlan's efficiency stems from the tight coupling between the pruning rules and the forward chaining search strategy. In addition, TLPlan allows for almost arbitrary complex control formulas that can generally be evaluated efficiently at each node (the process is in general intractable but in practice it appears efficient for control information Bacchus and Kabanza 1998). In the Graphplan or Blackbox framework the search proceeds very differently. The planning task is captured as a set of constraints mapped out over a fixed number of time steps. In Graphplan, the constraints are captured in a planning graph, which is subsequently searched for a valid plan. In Blackbox, the constraints are translated into a propositional formula (CNF), which can be searched with a satisfiability tester of the user's choice. In any case, in both Graphplan and Blackbox, the search does not proceed through a set of well-defined world states. In fact, the search may even involve as intermediate states that are unreachable from the initial state or even physically impossible. Especially in Satplan the search is difficult to characterize, because the problem is reduced to a generic propositional representation, without an explicit link to the original planning problem. The SAT solvers proceed in finding a truth assignment for the encoding (corresponding to a valid plan) without taking into account the fact that the encoding represents a planning problem (Baioletti *et al.* 1998; Ernst *et al.* 1997; Kautz and Selman 1998).

One way to incoporate the control knowledge from TLPlan into a Satplan encoding is by specifying additional constraints. We have extended the PDDL planning input language (McDermott 1998) of the Blackbox planner to allow for plan control as specified in temporal logic formulas.[1] The control knowledge is automatically translated into a class of additional propositinal clauses, which are added to the planning formula. We will discuss below the kinds of control that can be efficiently translated into a set of constraints, as well as the rules that cannot be captured efficiently.

Using a detailed experimental evaluation, we will show that control knowledge can indeed speed up the plan search significantly in the Blackbox framework. We also provide a detailed comparison with TLPlan. As we will see, our planner becomes competitive with the TLPlan performance. Initially, this was somewhat of a disappointment to us because we assumed that the Blackbox framework with control should be able to outperform TLPlan with the same control. However, a closer inspection of the results clarified the difference in approaches.

TLPlan is good at finding plans with a relatively few actions but it does not do well in domains that allow for parallel actions. In particular, we studied the logistics planning domain. This domain involves the task of delivering a set of packages to a number of different locations using one or more planes and vehicles. When several planes are available, one can find parallel plans, where different packages are moved using different planes simultaneously, allowing one to minimize the overall time span of the plan. Much of the combinatorics of the domain arises from the problem of finding good ways to interleave the movements of packages and planes. The sophisticated control in TLPlan will dramatically narrow the search. However, combined with the depth-first forward chaining strategy, the planner generates highly sequential plans. In fact, on the larger problems, the plan will use a single plane to deliver all packages. (Such plans can actually be found in polynomial time.) Blackbox, on the other hand, naturally searches for the shortest parallel plan because it operates by searching plans that fit within a certain number of time steps. So, although both planners with control find plans in comparable amounts time, Blackbox produces plans that have a much higher level of parallelism, thereby tackling the true combinatorial nature of the underlying planning problem. It is not clear how TLPlan can be made to generate more parallel plans. (We decribe several attemps to increase TLPplan's parallelism below.) Which planner should be preferred will depend on one's application and the level of inherent parallelism in the domain.

We believe our analysis provides new insights into the relative performance of different state-of-the-art planning methods. We hope that our findings can be used to further enhance these methods, and in general deepen our understanding of the design space of planning systems (Kambhampati 1997).

[1]Source code and data available from the first author.

Temporal Logic for Control

In TLPlan the control knowledge is encoded in temporal logic formulas (Bacchus and Kabanza 1996; 1998). The best way to introduce this approach is by considering an example control formula:

$$\Box\ (\forall\,[p : \mathit{airplane}(p)]\ \exists\,[l : \mathit{at}(p,\,l)]\ \forall\,[o : \mathit{in\_wrong\_city}(o,\,l)]\quad \mathit{in}(o,\,p) \Rightarrow \bigcirc\ \mathit{in}(o,\,p))$$

In temporal logic, $\Box f$ means f is true in all states from the current state on and $\bigcirc f$ means f is true in the next state. Therefore, the above formula can be read as "If a package o is in airplane p, p is at location l, and l is not in o's goal city, then package o should stay in airplane p in the *next* time step", and the above sentence should "always" hold. Predicate *in_wrong_city* is defined as follows:

$$\mathit{in\_wrong\_city}(o,l) \equiv \exists\,[g{:}\mathrm{GOAL}(\mathit{at}(o,g))]\ \exists\,[c : \mathit{in\text{-}city}(l,c)]\ \neg\mathit{in\text{-}city}(g,c)$$

After careful review of the control rules used by Bacchus and Kabanza, we found that the rules can be classified into the following categories:

I Control that involves only static information derivable from the initial state and goal state;

II Control that depends on the current state and can be captured using only static user-defined predicates;

III Control that depends on the current state and requires dynamic user-defined predicates.

The meaning of these categories may not be immediately obvious. Hopefully, the detailed examples below will clarify the distinctions. We will focus on two different ways of incoporating control: by pruning the planning graph (rules from category I), and by adding additional propositional clauses to the planning formula (rules from categories I and II). The rules in catergory III cannot be captured compactly.

Control by Pruning the Planning Graph

Graphplan constructs a special graph structured, called the planning graph, from the initial plan specification. Graphplan uses this graph to search for a plan leading from the intial state to the goal state. Blackbox translates the graph into a propositional encoding and then uses various SAT solution methods to search for a satisfying assignment, which corresponds to a plan. (This strategy closely mimics the original Satplan approach using linear encodings generated directly from a set of logical axiom schemas.) The planning graph contains two types of nodes, proposition nodes ("facts") and action nodes, arranged into levels indexed by time.

We can use a control formula to directly prune nodes in this planning graph. Consider the following control rule (category I) in the logistics domain.

Rule 1 Do not unload an object from an airplane unless the object is at its goal destination.

To illustrate this rule, suppose object `obj-1` is initially located in city `A` and its destination location is in city `C`. Once `obj-1` is loaded into an airplane `plane-1`, it is not necessary to unload `obj-1` at the airports in cities other than `C`. In other words, action `(UNLOAD-AIRPLANE obj-1 plane-1 B-airport)` can be removed from the planning graph at all levels (provided that B is another city). After pruning these nodes, one can also prune, facts nodes `(at obj-1 B-airport)`, since this cannot be achieved by any other action except for the one we just pruned.

In constructing the planning graph, the planner can be instructed to prune action nodes in the plan-graph as implied by the category I rules. In our implementation, rule #1 is captured by augmenting the original PDDL (Mcdermott 1998) planning language as follows:

```
(:action UNLOAD-AIRPLANE
  :parameters (?obj ?airplane ?loc)
  :precondition
     (and (obj ?obj) (airplane ?airplane)
          (location ?loc) (in ?obj ?airplane)
          (at ?airplane ?loc))
  :effect
     (and (not (in ?obj ?airplane))
          (at ?obj ?loc)))

(:defpredicate in_wrong_city
 :parameters (?obj ?loc)
 (exists (?goal_loc)
   (goal (at ?obj ?goal_loc))
      (exists (?city) (in-city ?loc ?city)
        (not (in-city ?goal_loc ?city)))))

(:action UNLOAD-AIRPLANE
 :exclude (in_wrong_city ?obj ?loc))
```

The user-defined predicate `in_wrong_city` is used to determine whether a location is the goal location for an object. It is worth noting that the value for predicate `in_wrong_city` can be decided purely based on the information in the goal state. The instantiated action `UNLOAD-AIRPLANE` will not be added to the planning graph when the predicate `(in_wrong_city ?obj ?loc)` is true. (Note the "exclude" condition in the `UNLOAD-AIRPLANE` rule.) The `in_wrong_city` predicate is purely an auxiliary predicate, it is not incorporated in the final planning graph.

Before an instantiated action is added into the graph, it will be checked against all exclude conditions for that action; if any of them holds, the action will be pruned and the effects introduced by that action at the next proposition node level may be pruned as well.

Table 1 shows the planning graph size before and after pruning. We see that, in the logistics domain, when applying the category I control rules from TLPlan, the number of nodes is reduced by ≈40%.

problem	length	#nodes	#nodes
		orig.	w. pruning
log-a	11	4246	2825
log-b	13	5177	3437
log-c	13	6321	3815
log-d	14	9842	5135

Table 1: The effect of graph pruning (category I rules).

This approach can be easily applied to the descendants of Graphplan and Blackbox. However, pruning of the planning graph does require that the rules rely solely on information of the goal state and possibly the initial state. Adding new propositional clauses provides a more powerful mechanism for adding control.

Control by Adding Constraints

For some domain-specific knowledge which cannot be captured via pruning of the planning graph, we can often add additional clauses to the propositional plan formulation as used in Blackbox to capture the control information. Consider the following control rule (category II):

Rule 2 Do not move an airplane if there is an object in the airplane that needs to be unloaded at that location.

First note that this rule cannot be captured using graph pruning. For example, consider removing the node `(FlY-AIRPLANE plane-1 city1 city2)` in layer i (*i.e.*, time i). Whether this can be done would depend on the truth value of the predicates that indicate what is in `plane-1` at time i, but of course, those truth values are not known in advance and are actually different for different plans. Therefore the node cannot be removed from the graph without the risk of losing certain plans. What we need to do is add clauses to the formula that in effect capture the rule but depend also on the truth values of the propositions that indicate what is in the plane.

Logically, the rule can be illustrated as follows:

$\forall\, pln, loc\, \forall[obj : \texttt{(not (in\_wrong\_city(}obj, loc\texttt{)))}]$
$(at(pln, loc, i) \wedge in(obj, pln, i)) \Rightarrow at(pln, loc, i+1)$

The following shows how to translate the above rule in our implementation:

```
(:wffctrl w3
 :scope
   (forall (?pln) (airplane ?pln)
      (forall (?loc) (airport ?loc)
         (forall (?obj) (obj ?obj)
            (not (in_wrong_city ?obj ?loc))
   )))
 :precondition
   (and (at ?pln ?loc) (in ?obj ?pln))
 :effect (next (at ?pln ?loc)))
```

The `:scope` field defines the domain where the rule applies, while `:precondition` and `:effect` fields represent the antecedent and consequent for the implication expression which will be added to the propositional formula. As mentioned earlier, predicate `in_wrong_city` is only used by the system and it will not be be added into the formula.

Similary, we can also translate rule #1 into propositional form:

```
(:wffctrl wg6
 :scope
   (forall (?pln) (airplane ?pln)
      (forall (?obj) (obj ?obj)
         (forall (?loc) (airport ?loc)
            (in_wrong_city ?obj ?loc)
   )))
 :precondition
   (and (at ?pln ?loc) (in ?obj ?pln))
 :effect (next (in ?obj ?pln)))
```

		before simpl.		after simpl.	
Problem	length	pruning	prop.	pruning	prop.
		#vars	#vars	#vars	#vars
rocket-a	7	1004	1337	826	826
rocket-a	7	1028	1413	868	868
log-a	11	2175	2709	1505	1511
log-b	13	2657	3287	2182	2182
log-c	13	3109	4197	2582	2590
log-d	14	4241	6151	3547	3551
log-e	15	5159	7818	4285	4285

Table 2: Comparison between planning graph pruning and propositional control of category I control rules.

One important observation about the application of the above rule is that the predicate `in_wrong_city` does not need to be added to the formula. It simply functions as a filter (indicated by the "`scope`" keyword) for adding the clauses defined by `:precondition` and `:effect` part. By doing so we do not lose any information because the `in_wrong_city` is static (independent of current state) and completely defined by the goal state. For example, if the goal destination of `package1` is in city `A`, it follows that `(in_wrong_city package1 B)` is true, independent of the current state. Below, we will see that category III rules involve defined predicates that do not have this property.

One reasonable question to consider is how adding rules of category I using additional clauses compares to the graph pruning approach discussed earlier. In the table 2, we consider the planning formulas as created from the planning graph. (We used the 6 category I control rules from TLPlan.) The first two columns give the number of variables for the two different strategies. As might be expected, planning graph pruning strategy gives the smallest number of variables. However, we also included the result of running a polynomial time simplification procedure (Crawford 1994). As can be seen from the last two columns, the remaining numbers of variables are identical for most formulas, with only some small differences for certain instances. When we checked into the remaining differences, we found that those result from some specialized additional pruning Blackbox does specifically for the final layer of the planning graph. Overall, table 2 shows that direct graph pruning or a coding via additional clauses leads to formulas on essentially the same set of variables. (We also verified that the variables actually refer to the same planning propositions in terms of the original planning problem.) As a result, the two mechanisms are essentially equivalent, although they do capture the planning problem using a different clause set. Below we will compare the computational properties of these two approaches.

Rules With No Compact Encoding

We now consider the category III rules. Although these rules can be translated into additional propositional constraints in principle, they do lead to an impractical number of large clauses to be added to the formula. Consider the following rule from TLPlan:

Rule 3 Do not move a vehicle to a location unless, (1) the location is where we want the vehicle to be in the goal, (2) there is an object at that location that needs to be picked up, or (3) there is an object in the vehicle that needs to be unloaded at that location.

The main purpose of rule #3 is to avoid unnecessary move of vehicles. First of all, the rule requires current state information, and therefore cannot be handled by graph pruning. Secondly, in order to translate the rule into propositional constraints, we need to introduce extra predicates to represent the idea that there is an object in the destination location that needs to be loaded or unloaded by the vehicle. For example, in order to avoid unnecessary move of airplanes (a form of vehicle), one way to encode it is to define predicate *need_to_move_by_airplane* for each airport first:

$\forall obj, \forall[apt : \texttt{in\_wrong\_city(obj,apt)}]$
$at(obj, apt, i) \Rightarrow need\_to\_move\_by\_airplane(apt, i)$

$\forall apt\ need\_to\_move\_by\_airplane(apt, i) \Rightarrow$
$\exists[obj : \texttt{in\_wrong\_city(obj,apt)}]\, at(obj, apt, i)$

		blackbox	blackbox(I_a)	blackbox(I_b)	blackbox(II)	blackbox(I_a&II)	blackbox(I_b&II)
Problem	length	time	time	time	time	time	time
rocket-a	7	2.06	4.20	3.41	2.10	3.92	3.82
rocket-b	7	2.87	2.09	2.46	3.26	1.85	2.49
logistics-a	11	3.80	2.78	3.64	3.74	2.78	3.63
logistics-b	13	4.83	3.46	4.56	4.75	3.41	4.66
logistics-c	13	6.75	3.89	5.64	6.71	3.94	5.81
logistics-d	14	15.85	7.29	10.4	15.69	6.82	10.23
logistics-e	15	3522	151	201	2553	60	148
logistics-1	9	4.80	3.68	4.97	4.83	3.70	4.98
logistics-2	11	406	>7200	270	360	130	141
tire-a	12	1.37	1.34	1.35	1.36	1.33	1.34
tire-b	30	114	93	72	55	21	23

Table 3: Blackbox with control knowledge. Experiments were run on a 300Mhz Sparc Ultra. Times are given in cpu seconds.

I_a: Category I control knowledge used for pruning planning graph.
I_b: Category I control knowledge added in propositional form.
II: Category II control knowledge added in propositional form.

Similarly, we can also define an extra predicate *need_to_unload_by_airplane*. And finally, we will need the following to translate the rule:

$$\forall pln, \forall[apt1, apt2 : \texttt{(not (= apt1 apt2))}]$$
$$(at(pln, apt1, i) \wedge$$
$$\neg need\_to\_unload\_by\_airplane(apt2, i) \wedge$$
$$\neg need\_to\_move\_by\_airplane(apt2, i))$$
$$\Rightarrow \neg at(pln, apt2, i+1)$$

Suppose there are n objects, m cities, and k airplanes. The encoding will introduce $O(mn)$ predicates and $O(mn + km^2)$ propositional clauses in each time step. Furthermore, some of them are lengthy clauses, containing up to mn number of literals. We explored adding this kind of control information to our formulas but, except for the smallest planning problems, the formulas become too large for our SAT solvers. The key difference with a category II rule (such as rule #2) is that in this case we also need to add clauses that capture our defined predicates, such as *need_to_move_by_airplane*. This is in contrast with the encoding of, *e.g.*, rule #2, where, because of the static nature of the defined predicates, they can simply be used as a filter when adding clauses (see discussion on `:scope` above). This cannot be done for our *need_to_move_by_airplane* because its truth value depends on where a package is at the current time. As a consequence, the category III rules are examples of rules that cannot effectively be captured into a constraint-based planner, such as Blackbox. Fortunately, as we will see below, in terms of computational efficiency the category I and II rules appear to do most of the work, at least in the domains we considered. An interesting research question is whether there is a more effective way to encode rules such as rule #3.

Empirical Evaluation

The testbed used in this paper is a series of problems from the logistics planning domain and the rocket domain (Veloso 1989; Blum and Furst 1995; Selman and Kautz 1996; Mcdermott 1998), and the tire-world domain from the TLPlan distribution (Bacchus and Kabanza 1998). In addition, we created two new problem instances: logistics-1 and logistics-2, which can be solved with highly parallel plans (up to 20 actions in parallel).

Table 3 gives the result of Blackbox running on problems with different levels of domain knowledge. The column labeled “Blackbox” gives the runtime without any control knowledge. Subsequent columns give results for the different control strategies. (Results of randomized solvers were averaged over 10 runs. We used 6 category I rules and 5 category II rules.) As a general observation, we note that the effect of control knowledge becomes more apparent for the larger problem instances. For example, on our hardest problem, “logistics-e”, basic Blackbox takes almost one hour, but with control (catergory I_a & II) it only takes 60 seconds. The results on the larger instances are more significant than those for the smaller problems, because the solution times on the smaller instances are often dominated by basic I/O operations such as reading the formula from disk, as opposed to the actual compute time for solving the formulas.

Based on the table, we can make the following observations:

- Control information does reduce the solution time, especially on the harder problem instances. Consider the data on logistics-e, logistics-2, and tire-b.
- Control category I rules, encoded via graph pruning and as additional clauses (columns I_a &I_b) lead to roughly the same speedup on the larger problems. One notable exception is the logistics-2 problem, where the solution time actually goes up for strategy I_a. This is most likely a consequence of the fact that clauses are eliminated by the pruning, lead-

	tlplan-dfs			blackbox(I_a&II)			tlplan-rand-dfs		
problem	length	#action	time	length	#action	time	length	#action	time
logistics-a	13	51	0.49	11	72	2.78	15	57	0.66
logistics-b	15	42	0.4	13	71	3.41	15	46	0.43
logistics-c	17	51	0.64	13	83	3.94	15	55	0.72
logistics-d	26	70	2.13	14	104	6.82	18	75	2.43
logistics-e	24	89	4.27	15	107	60	23	96	4.57
logistics-1	15	44	0.9	9	77	3.70	15	49	1.01
logistics-2	29	93	33.16	11	147	130	16	100	34.52

Table 4: Comparison between TLPlan and Blackbox.

ing to less propagation in the SAT solver. We do not see this problem when the knowledge is added via additional clauses (I_b), which therefore appears to be a more robust strategy. The smaller problems instances are already solved within a few seconds by basic Blackbox: not much can be gained from control (again, partly because I/O dominates the overall times).

- Category I rules are most effective on problems from the logistics domain; category II rules are more effective in the tireworld. One interesting research issues is whether one can identify in advance which kinds of rules are most effective. (Measurements such as clause-to-variable ratios and degree of unit propagation may be useful here.)
- The effect of control is largely cumulative. Our category I rules combined with category II lead to the best overall performance (see columns I_a&II and I_b&II).

Next, we compare the performance of TLPlan and Blackbox. Table 4 gives our results. We note that, in general, TLPlan is still somewhat faster than Blackbox with similar control. Note that both planners now use the same control information except for some category III rules in TLPlan. However, the differences are much smaller than with the original Blackbox without control (Table 3 and Bacchus and Kabanza 1998). We believe these results demonstrate that we can meet the challenge, proposed by Bacchus and Kabanza, to effectively incorporate declarative control as used in TLPlan into a constraint-based planner. Nevertheless, as noted in the introduction, we were somewhat disappointed that Blackbox with control was not faster than the TLPlan approach, given the more sophisticated search of the SAT solvers. In order to get a better understanding of the issues involved, we consider the plan quality of the generated plans.

Table 4 gives plan length in terms of number of actions and parallel time steps for the logistics domain. Note that the logistics domain allows for a substantial amount of parallelism because several planes can fly simultaneously. We see that TLPlan often finds plans with fewer actions; however, in term of parallel lengths the plans are much longer than those found by Blackbox. In fact, Blackbox because of its plan representation can often find the minimal length parallel plan. Especially on the larger problems, we observe a substantial difference. For example, on logistics-e, TLPlan requires 24 time steps versus 15 for Blackbox. After a closer look at the plans generated by TLPlan, we found plans that only use a single plane to transport the packages. Such plans can be found very fast (polynomial time) but ignore much of the the inherent combinatorics of the domain. The reason TLPlan finds such plans is a consequence of its control rules and the depth-first search strategy.

It is not clear how one can improve the (parallel) quality of the plans generated by TLPlan. Part of the difficulty lies in the fact that the control knowledge is taylored towards more sequential plans. For example, rules that keep a plane from flying if there are still packages to be picked up at a location. (This prevents a plan where another plane picks up the package later.) In addition, the depth-first style search also tends to steer the planner towards more sequential plans (*e.g.,* always picking plane-1 to move). We experimented with several different search strategies to try to improve the plans generated by TLPlan. First, we checked wether the parallel quality of plans obtained with breadth first search was better. This does not appear to be the case. (We could only check this on very small instances, because the search quickly runs out of memory.) An approach that does lead to some improvement is to randomize the depth-first search. Table 4 shows some improvement in parallel length but still not close to the minimal possible. The reason for the improvement is that TLPlan now can pick different planes more easily in its branching. (The deterministic depth-first search repeatedly selects planes in the same order.) It's an interesting question how TLPlan can be made to find more parallel plans and how this would affect its performance.

Conclusions

Intuitively speaking, the control in TLPlan attempts to push the planning problem into a polynomial solvable problem. This is along the lines of the general focus in planning on eliminating or avoiding search as much as possible.[2] TLPlan achieves its objective in a very elegant manner because the control rules are quite intuitive and purely declarative. In combination with the forward chaining planner, the rules do lead to polynomial scaling in a number of interesting domains. Our analysis shows that there is a price to be paid for this gain in efficiency, which is the loss of much of the parallel nature of many planning tasks.[3]

We have shown that one can obtain the benefits of the control rules in terms of efficiency, without paying the price of reduced parallelism, by incorporating the control rules in the Blackbox planner. In a sense, the SAT solvers in Blackbox still tackle the combinatorial aspect of the task but the extra constraints provide substantial additional pruning of the search space.

We implemented the system by enhancing the Blackbox planner with a parser for temporal logic control rules, which are translated in additional propositional clauses. We also showed that a subset of the control (category I rules) can also be handled by direct pruning of the planning graph. Our experimental results show a speedup due to the search control of up to order of magnitude on our problem domains. Given the effectiveness of the control, it would be interesting to add further constraints, such as state invariants (Fox and Long 1998; Gerevini and Schubert 1998; Kautz and Selman 1998).

We believe our work demonstrates that declarative control knowledge can be used effectively in constraint-planners without loss of (parallel) plan quality. Compared to TLPlan, which is a highly efficient planner in and of itself, the main advantage of our approach is that we maintain parallel plan quality. Overall, incorporating declarative control in constraint-based planning appears to be a promising research direction. With more sophisticated control, another order of magnitude speedup may be achievable.

Another fascinating direction for future research is the possible use of rule-based learning techniques for acquiring control knowledge automatically by "training" the planner on a sequence of smaller problems. Learning of control knowledge has been explored previously for other, more procedural, planners. See, for example, Etzioni (1993), Knoblock (1994), Minton (1988), and Veloso(1992). We are currently exploring forms of control rule learning in our declarative constraint-based framework.

Acknowledgements We thank Fahiem Bacchus, Carla Gomes, Dana Nau, and Dan Weld for many useful comments and suggestions. The second author received support from by an NSF Career grant and a Sloan Fellowship.

[2] Austin Tate is said to have said "If you need to search, you're dead."

[3] Given that these planning problems are NP-complete, it is clear that something will be lost by solving them in polynomial time.

References

Bacchus, F. and Kabanza, F. (1996). Using temporal logic to control search in a forward-chaining planner. In *New Dir. in Planning*, M. Ghallab and A. Milani (Eds.), IOS Press.

Bacchus, F. and Kabanza, F. (1998). Using Temporal Logics to Express Search Control Knowledge for Planning. See http://www.lpaig.uwaterloo.ca/~fbacchus/on-line.html.

Baioletti, M. , Marcugini, S., and Milani, A. (1998). C-SATPlan: a SATPlan-based tool for planning with constraints. AIPS-98 Workshop on Planning as Combinatorial Search, Pittsburgh, PA.

Blum, A. and Furst, M.L. (1995). Fast planning through planning graph analysis. *Proc. IJCAI-95*, Canada.

Crawford, C. (1984). Compact: A fast simplifier of Boolean formulas. Available via Crawford's web page.

Ernst, M.D., Millstein, T.D., and Weld, D.S. (1997). Automatic SAT-compilation of planning problems. *Proc. IJCAI-97*, Nagoya, Japan.

Etzioni, Oren (1993). Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2), 255-302.

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 5(2): 189-208.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. Forthcoming.

Geffner, H. (1998). HSP: Heuristic Search Planner, Working notes of the Workshop on Planning as Combinatorial Search, Pittsburgh, PA, 1998.

Gerevini, A. and Schubert, L. 1998. Inferring state constraints for domain-independent planning. *Proc AAAI-98*, Madison, WI.

Kambhampati, S. (1997). Challenges in bridging plan synthesis paradigms. *Proc. IJCAI-97*, Nagoya, Japan.

Kautz, H. and Selman, B. (1992). Planning as satisfiability. *Proc. ECAI-92,* Vienna, Austria, 359–363.

Kautz, H. and Selman, B. (1996). Pushing the envelope: planning, propositional logic, and stochastic search. *Proc. AAAI-1996,* Portand, OR.

Kautz, H. and Selman, B. (1998). The role of domain-specific axioms in the planning as satisfiability framework. *Proc. AIPS-98,* Pittsburgh, PA.

Kautz, H. and Selman, B. (1999). Unifying SAT-based and Graph-based planning. Proc. IJCAI-99, to appear.

Knoblock, C. (1994). Automatically generating abstractions for planning. *Artificial Intelligence* 68(2).

Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. *Proc. 4th European Conf. on Planning*, S. Steel, ed., vol. 1248 of *LNAI*, Springer.

McDermott, D., *et al.* (1998). PDDL — the planning domain definition language. Draft.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. *Proc. AAAI-88*, St. Paul, MN, 564-569.

Veloso, M. (1992). Learning by analogical reasoning in general problem solving. Ph.D. Thesis, CMU, CS Techn. Report CMU-CS-92-174.

Weld, D. *et al.* (1999). Recent advances in AI planning. AI Magazine (to appear). Available from author's web site.