Linkability: Examining Causal Link Commitments in Partial-Order Planning*

Manuela Veloso and **Jim Blythe**
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3890
{veloso,blythe}@cs.cmu.edu

Abstract

Recently, several researchers have demonstrated domains where partially-ordered planners outperform totally-ordered planners. In (Barrett & Weld 1994), Barrett and Weld build a series of artificial domains exploring the concepts of trivial and laborious serializability, in which a partially-ordered planner, SNLP, consistently outperforms two totally-ordered planners. In this paper, we demonstrate that totally-ordered planners sometimes have an advantage over partially-ordered planners. We describe a series of domains in which PRODIGY4.0 consistently outperforms SNLP, and introduce the concept of linkability to characterize the class of domains for which this happens. Linkability highlights the fact that partially-ordered planners commit to causal links in much the same way that totally-ordered planners commit to step ordering.

Introduction

Recently, several researchers have advocated the view that partially-ordered planners are generally more efficient than totally-ordered planners (Barrett & Weld 1994; Kambhampati & Chen 1993; Minton, Bresina, & Drummond 1991). Barrett and Weld in particular (Barrett & Weld 1994) built a series of artificial domains on which a partially-ordered planner, SNLP, consistently outperforms two totally-ordered planners. The domains are built to examine some refinements to the concept of *serializability* introduced by Korf (Korf 1987), and to highlight their role in predicting the performance of planners.

We examine this view by comparing PRODIGY4.0, a totally-ordered planner, with SNLP. PRODIGY4.0 (Carbonell & the PRODIGY Research Group 1992) is based on PRODIGY2.0 (Minton *et al.* 1989) and NOLIMIT (Veloso 1989). Like the two totally-ordered planners used in (Barrett & Weld 1994), it is complete, solving the same class of problems as SNLP. PRODIGY4.0 commits to a total order of operators at each stage in its search, and backtracks when interactions are detected. PRODIGY4.0 uses a suite of machine learning modules to learn from experience in planning in a particular domain.

We based our investigations of the claims of (Barrett & Weld 1994) on two observations. First, PRODIGY4.0 commits to a total order of plan steps in order to make use of a uniquely specified world state while planning. Second, any system that treats planning as a search process will make a series of commitments during search. The pattern of commitments made will produce greater efficiency in some kinds of domains, and less in others.

So far, there has not been a clear demonstration of the kinds of domains where totally-ordered planners may perform better than partially-ordered planners. In order to address this, we designed a set of artificial planning problems in which we carefully exploit the use of a world state in the planning process. In these domains PRODIGY4.0 consistently outperforms SNLP. The domains show how reasoning about the state guides the planner to select efficient planning orderings of goals and correct operators. The domains also highlight the fact that although partially-ordered planners like SNLP use a "least-commitment" approach to step ordering, they commit eagerly to causal links, which can be inefficient. Based on the characteristics of these domains, we introduce the concept of "linkability", which we compare with serializability.

Although the domains we describe here show PRODIGY4.0 consistently outperforming SNLP, it is not our aim to prove that one or another planning strategy is the best in all domains. Rather, we are highlighting the fact that different strategies may have analogous problems with different domains. We believe that there is no universally superior strategy of search commitments for planning, and this leads us to conclude that one of the most important research issues in domain-independent planning is to learn a mapping

*This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Wright Laboratory or the U. S. Government.

from measurable characteristics of problems into the most appropriate planning strategy.

The layout of the paper is as follows. In the next section we briefly present the planning algorithms used by PRODIGY4.0 and SNLP. Then we introduce the concept of "linkability," and present the domains we created that display different levels of linkability. We show the empirical results we obtained illustrating that all these domains are trivially linkable for PRODIGY, while not for SNLP. In the final section we draw conclusions from this work. The reader familiar with the PRODIGY and SNLP planners may skip the next section without loss of content.

Brief discussion of PRODIGY and SNLP

PRODIGY is an integrated architecture for problem solving and learning. It consists of a core planning algorithm whose knowledge base and planning behavior are available for several machine learning modules to interpret and compile planning expertise.

PRODIGY's planning algorithm interleaves backward-chaining planning with the simulation of plan execution, by applying operators found relevant to the goal to an internal world state. Although there are some recognized situations where plain state-space search is not efficient, the advantage of considering the simulated state of the world while planning has not been clearly shown. Table 1 shows an abstract view of PRODIGY's planning algorithm.

1. Compute the set $\mathcal{G}$ of goal literals left to be achieved: if $\mathcal{S}$ is the set of literals in the current state, and $\mathcal{P}$ is the set of all the preconditions of the set $\mathcal{O}$ of operators that have been selected, then $\mathcal{G} = \mathcal{P} - \mathcal{S}$.
2. Terminate if $\mathcal{G}$ is empty. Return the sequence of operators that were applied as the plan.
3. Choose a goal from $\mathcal{G}$ to plan for.
4. Choose an operator and bindings to achieve the selected goal. If all of the preconditions of this operator are true in the current state, apply it to the current state and get a new state. Otherwise add the operator to $\mathcal{O}$ and go to step 1.
5. Recursively apply any operator in $\mathcal{O}$ that becomes applicable in the new state and remove it from $\mathcal{O}$.
6. Go to step 1.

Table 1: Abstract view of PRODIGY*'s planning algorithm*

This is a simple version of PRODIGY's planning strategy. PRODIGY uses several domain-independent and domain-dependent search heuristics that extend this basic search procedure, and choices are then based on more sophisticated analysis of the planning state (Stone, Veloso, & Blythe 1994). For the purpose of this paper, however, the simple version with its eager application strategy is adequate to show the effect of planning using the simulated execution state. PRODIGY considers different operator orderings if needed, by backtracking over the simulated plan execution.

SNLP is an example of a planner that searches in the plan space. It is viewed as following a "least-commitment" planning strategy. It does not commit to a particular ordering of operators nor does it keep a current state of the world.

Table 2 shows an abstract view of SNLP's planning algorithm (McAllester & Rosenblitt 1991).

1. Terminate if the goal set is empty.
2. Select a goal g from the goal set and identify the plan step that needs it, S_{need}.
3. Let S_{add} be a step that adds g, either a new step or a step that is already in the plan. Add the causal link $S_{add} \xrightarrow{g} S_{need}$, constrain S_{add} to come before S_{need}, and enforce bindings that make S_{add} add g.
4. Update the goal set with all the preconditions of the step S_{add}, and delete g.
5. Identify *threats* and resolve the conflicts by adding ordering or bindings constraints.
6. Go to step 1.

Table 2: Abstract view of SNLP*'s planning algorithm*

SNLP does indeed commit less than PRODIGY in terms of its operator orderings, since it does not need to commit to change the state as PRODIGY does. This means it is unaffected by problems created by choosing a poor goal ordering which affect PRODIGY. However we show in this paper that its commitment to a causal link for each goal can be as problematic as the operator ordering commitments in PRODIGY.

Linkability

Our definition of "linkability" is somewhat analogous to the definition of "serializability", introduced by Korf (Korf 1987):

> A set of subgoals is *serializable* if there exists some ordering whereby they can be solved sequentially without ever violating a previously solved subgoal.

Here, a "subgoal" is a property of states, formally defined as a set of states. In (Barrett & Weld 1994), Barrett and Weld identify subgoals with sets of "plan states." A plan state specifies a set of steps and ordering and bindings constraints on them, and is an intermediate state for a plan-space planner such as SNLP. Notice, however, that a plan state ignores the set of causal links. Barrett and Weld go on to make two refinements to the concept of serializability, *trivial* and *laborious* serializability, which refer to the proportion of orderings which can be solved sequentially. The intuition behind these refinements is that the planner must backtrack when it picks an ordering that cannot be solved sequentially, and the combinatorial effect of this will dominate the time taken by the planner on average as long as some small proportion of orderings

Goal: `polished` and `has-hole` Initial state: *empty* PRODIGY	Goal: `polished` and `has-hole` Initial state: `polished` SNLP
plan for goal `polished` *select operator* `Polish` • *order* `Polish` *as first operator in plan* *plan for goal* `has-hole` *select operator* `Drill-Hole` • *order* `Drill-Hole` *after* `Polish` • `polished` *is deleted, backtrack* *order* `Drill-Hole` *as first operator* *order* `Polish` *after* `Drill-Hole`	*plan for goal* `polished` *select operator* `initial state` • *link* `initial state` *to* `polished` *plan for goal* `has-hole` *select operator* `Drill-Hole` • *link operator* `Drill-Hole` *to* `has-hole` • *threat* - `polished` *needs to be relinked* *select operator* `Polish` *link operator* `Polish` *to* `polished` *order* `Drill-Hole` *before* `Polish`

Figure 1: An example of the parallel between linking and ordering commitments. Notice the difference of initial state in the two situations.

require backtracking. Thus, they provide better indicators of planner performance.

While serializability captures problem complexity due to interactions between subgoals, it does not account for all the choice points where planners have to make commitments. The choice in SNLP's algorithm of which step to use to achieve an open goal is sometimes independent of goal ordering, as is the related choice of operator in PRODIGY.

Consider the following example to motivate the concept of linkability. Let a domain consist of the following two operators:

```
Operator Polish            Operator Drill-Hole
 preconds: ()               preconds: ()
 adds:     polished         adds:     has-hole
 deletes:  ()               deletes:  polished
```

Suppose that the two goals `polished`, and `has-hole` are given to a planner and that neither goal is satisfied in the initial state. PRODIGY may commit to the wrong operator ordering, first trying the plan `Polish - Drill-Hole` and then backtracking to the correct plan `Drill-Hole - Polish`. On the other hand, SNLP directly finds the solution independently of the order in which it addresses the two goals.

Now consider the situation in which the literal `polished` is satisfied in the initial state. PRODIGY takes the state into account when deciding which goals it needs to plan for. Therefore, it starts correctly planning for the goal `has-hole`. On the other hand, if SNLP starts planning for the goal `polished`, it will link it incorrectly to the initial state and later it will have to resolve a threat by backtracking over a previously established causal link.

Figure 1 illustrates this parallel between the two commitment strategies. It can be seen that they lead to surprisingly similar problems.

Leading from this example, *linkability* in partial order planners focuses on the need to backtrack from commitments made to causal links to achieve goals. Linkability therefore applied to causal link commitments would be stated as:

> A set of subgoals is *easily linkable* if, independent of the order by which the planner assigns causal links to the subgoals, it never has to backtrack to undo those links. Otherwise it is *laboriously linkable*.

Given a particular planning algorithm, the degree of linkability of a planning problem corresponds to the expected proportion of commitments that the planner will have to backtrack over.

A series of domains to explore linkability

We created a series of artificial domains to explore the linking commitments of SNLP. We believe linking commitments are common to all least-commitment or partial-order planners. Notice that these domains highlight the general problem that plan-space planners may face in choosing particular goal orderings and particular operators for a goal because they cannot reason about a precise world state while planning. We show three such domains and the results of running SNLP and PRODIGY4.0 in a series of problems in these domains.[1] We also briefly discuss some characteristics of these domains that we believe are representative of real world planning problems.

In our presentation in this section, the domains are ordered in an increasing level of difficulty of linkability.

[1] We thank Tony Barrett and Dan Weld for providing us with the code for SNLP.

Easily linkable goals

Consider a domain with n operators A_i each with an empty set of preconditions and deletes. A_i adds goal g_i. The domain has also an operator, A_* that has an empty set of preconditions, adds the goal g_* and deletes all of the goals g_i. We name this domain *link-simple*. Figure 2 shows the general operator A_i and the operator A_* in the domain.

operator A_i		operator A_*	
preconds	()	preconds	()
adds	g_i	adds	g_*
deletes	()	deletes	$g_i, \forall i$

Figure 2: The operators in the domain link-simple

For our experiments we used *link-simple* with $n = 15$. We generated a series of 15 classes of problems, each with 10 problems, giving a total of 150 problems. A problem in class k has goals g_* and goals g_1 through g_k randomly permuted. All the literals g_1 through g_k are present in the initial state. The 10 problems in each class correspond to different random permutations both of the initial conditions and goal literals. Figure 3 shows a problem from the 5th class, with g_* and five more goals in the goal statement and initial state. We also show a solution plan.

Initial state: g_1, g_2, g_3, g_5, g_4
Goal statement: $g_2, g_5, g_4, g_*, g_3, g_1$
Plan: $A_*, A_2, A_5, A_4, A_3, A_1$

Figure 3: A problem and sample solution for the domain link-0d-star

Figure 4 shows the results obtained running PRODIGY and SNLP, plotting average running time against the number of goals, showing error bars for the 90% confidence interval. For the sake of clarity, only 3 intervals are shown, as all the others are of identical size.

PRODIGY uses its current state to determine which goal literals need to be achieved following a means-ends analysis procedure. Therefore PRODIGY starts immediately planning for goal g_* which in this case is the only goal that is not satisfied in the initial state. The operator A_* is selected and applied to the initial state deleting all the g_i goals. As these goals are all independent, PRODIGY solves the problems in linear time in the number of g_i goals. In contrast, SNLP starts working on the goals in the order given in the goal statement. Each goal g_i encountered before g_* is mistakenly linked to the initial state. When SNLP plans for g_*, it finds the threats between A_* and the goals g_i that A_* deletes. It needs to solve these threats by undoing the causal links that it has introduced. SNLP also solves the problems

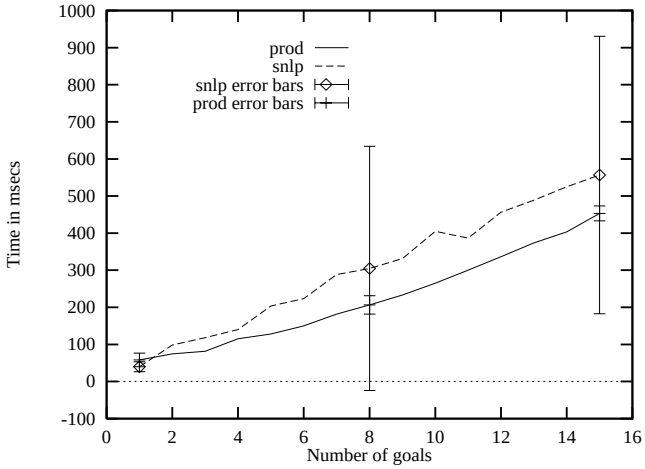

Figure 4: Average running time of PRODIGY4.0 *and* SNLP *in the domain link-simple. Errors bars for the 90% confidence interval are shown for 3 representative points.*

in linear time, but has to backtrack over all the links created before it encounters g_*.

The domain *link-simple* is our simplest example of the effects of linking commitments in planning performance. We introduce two additional domains where uninformed linking commitments create more severe problems.

Laboriously linkable

In this domain we explore the effect of having to repeat actions in a plan. We name this domain *link-repeat*. Figure 5 shows that each operator A_i needs as preconditions both literals g_* and g_{i-1}, adds the literal g_i, and deletes g_*. The additional operator A_* is always available to add g_* with no other effects.

operator A_i		operator A_*	
preconds	g_*, g_{i-1}	preconds	()
adds	g_i	adds	g_*
deletes	g_*	deletes	()

Figure 5: The operators in the domain link-repeat

We again used up to 15 goals g_i in our experiments. Each problem in this domain only has two goals, namely goal g_* and goal g_k, where k ranges from 1 to 15. Thus there are 15 problems in our test set. The initial state consists simply of the literal g_*. Figure 6 shows a problem and the corresponding plan.

The goal g_* needs to be reachieved each time one of the g_i goals is achieved, as all the operators A_is delete g_*. In addition, each goal g_i needs to be achieved before the goal g_{i+1}, as each g_i is a precondition to the operator A_{i+1}.

In problems in this domain, PRODIGY only encounters one goal at a time to plan for, when computing the difference between the needed preconditions and

Initial state: g_*
Goal statement: g_*, g_5
Plan: $A_1, A_*, A_2, A_*, A_3, A_*, A_4, A_*, A_5$

Figure 6: A problem and corresponding solution for the domain link-repeat

the world state. Its use of the state focuses PRODIGY on the correct goal to plan for and PRODIGY proceeds smoothly by adding planning steps to the state as they become needed. In this situation, SNLP suffers again from its linking eagerness biased towards linking new literals to existing operators in the plan, leading to costly backtracking.

Figure 7 shows the results obtained running PRODIGY and SNLP on the set of problems with 90% confidence intervals. In this graph we plot the average running time against the number k of the goal given in the problem, which also corresponds to the length of the solution.

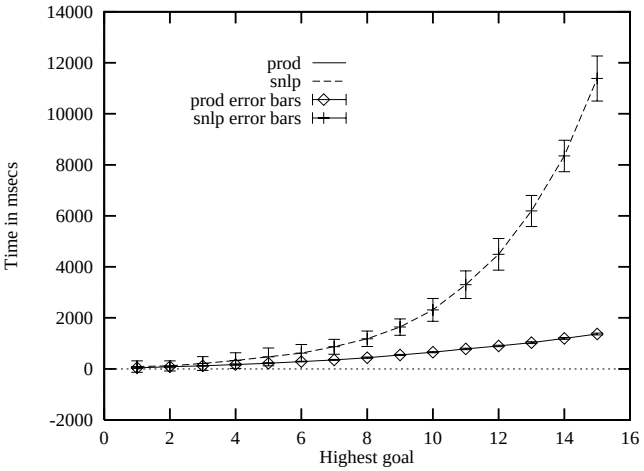

Figure 7: Average running time of PRODIGY4.0 *and* SNLP *in the domain link-repeat. Errors bars show the 90% confidence intervals.*

This domain is a simple instance of a real world class of domains where there is a limited number of resources that are consumed and can be reachieved while planning. The goal g_* in this domain characterizes as a resource such as fuel in a transportation domain, that may be used up and replenished. Similarly, in this domain, g_* needs to be reachieved every other step. In the general case, there may be any number of goals that need to be reachieved after any number of steps.

Multiple linking alternatives

Figure 8 shows a domain where each operator A_i needs as preconditions all the literals $g_j, \forall j < i$, and adds all the literals g_i except g_{i-1}, which is deleted. Figure 9 illustrates three operators of this domain.

operator A_i
preconds $g_j, \forall j < i$
adds $g_i, g_j, \forall j < i-1$
deletes g_{i-1}

Figure 8: The operators in the domain link-chain

operator A_5	operator A_4	operator A_3
pre g_*, g_4, g_3, g_2, g_1	pre g_*, g_3, g_2, g_1	pre g_*, g_2, g_1
add g_5, g_3, g_2, g_1	add g_4, g_2, g_1	add g_3, g_1
del g_4	del g_3	del g_2

Figure 9: Example of three operators in the domain link-chain

This domain is interesting as it illustrates a situation where there are several operators that can achieve the same goal. For example, the operators A_3 and $A_j, j \geq 5$ achieve the goal g_3. A_3 is the only correct choice, as the other operators need g_3 at the same time that they add it. This domain may seem rather artificial in the sense that the operators add their own preconditions. However one might get the same behavior from a combination of several operators in realistic domains.

We again used $n = 15$ in our experiments and generated 150 problems in a similar way to the link-simple domain. A problem in class k has k goals, g_1 through g_k, randomly permuted. The literals g_1 through g_{k-1} are present in the initial state. Figure 10 shows an example problem and the corresponding solution plan.

Initial state: g_1, g_4, g_3, g_2
Goal statement: g_2, g_5, g_4, g_3, g_1
Plan: A_5, A_4, A_3, A_2, A_1

Figure 10: A problem and the corresponding solution for the domain link-chain

Figure 11 shows the results obtained running PRODIGY and SNLP.

SNLP finds a large number of possible ways of linking each goal. The results show again that its linking strategy is inappropriate for this domain. Note that this domain is designed so that PRODIGY encounters only one goal at each time to plan for. Although there are multiple operators that add the same goal, once again PRODIGY uses the state and chooses the one that is immediately applicable and therefore PRODIGY finds the solution without backtracking.

It has been noticed before that partial-order planners, including SNLP may be inefficient when multiple operator choices are present (e.g. (Knoblock & Yang 1993)). The argument has been that state-space planners would necessarily find equivalent problems. However this is not the case in many situations in which the state may provide information which can be used by the total-order planners and not by the partial-order planners. This domain illustrates one such situation.

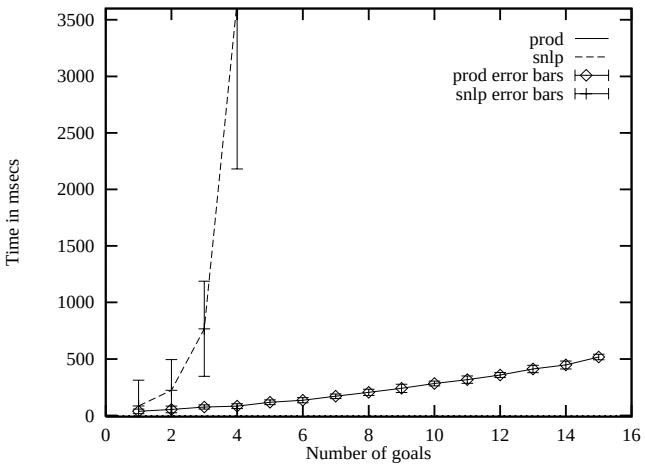

Figure 11: Average running time of PRODIGY4.0 *and* SNLP *in the domain link-chain. Errors bars show the 90% confidence intervals.*

Conclusion

In this paper we introduce the notion of linkability which highlights the fact that partially-ordered planners commit to causal links in much the same way that totally-ordered planners commit to step ordering. We created a series of artificial domains, where PRODIGY, a totally-ordered planner, consistently outperforms SNLP.

Taken with the results reported in (Barrett & Weld 1994), these results begin to delimit the types of domain for which the different commitment strategies are suitable. We conclude that there is no strategy of commitment that outperforms all others in all planning problems, and that a more fruitful line of research is to seek a structural model of planning problems that can predict the appropriate planning strategy. In particular, the description of a planning strategy as "least-commitment" without further qualification is somewhat misleading, since search itself is an iterative process of making a commitment and evaluating the commitment. Levels of commitment must therefore be viewed as a tradeoff across the different dimensions of a search problem. We have illustrated this tradeoff in the two planners considered in this paper.

We believe that the ideal planning algorithm would use a more flexible commitment strategy than is evident in the planners of today, accumulating information on which to base its next commitment before fixing the form of the commitment. Under different circumstances this algorithm might resemble a partially ordered planner that maintains least commitment to operator orderings, or a totally ordered planner that can make use of an internal world state. We are currently designing a planning algorithm based on this view.

Acknowledgements

We are grateful to members of the Prodigy research group for useful discussions about this work, and also thank Dan Weld and Tony Barrett for making their code available. The comments of two anonymous reviewers were very helpful in improving this paper. To obtain a copy of PRODIGY4.0, send electronic mail to prodigy@cs.cmu.edu.

References

Barrett, A., and Weld, D. S. 1994. Partial-order planning:Evaluating possible efficiency gains. *Artificial Intelligence* 67(1).

Carbonell, J. G., and the PRODIGY Research Group. 1992. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, School of Computer Science, Carnegie Mellon University. Jaime G. Carbonell and and the PRODIGY Research Group.

Kambhampati, S., and Chen, J. 1993. Relative utility of EBG based plan reuse in partial ordering vs. total ordering planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 514–519.

Knoblock, C. A., and Yang, Q. 1993. A comparison of SNLP and TWEAK planning algorithms. In *Working notes of the AAAI Spring Symposium on Foundations of Automatic Planning: The classical approach and beyond*, 73–77.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33:65–88.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 634–639.

Minton, S.; Knoblock, C. A.; Kuokka, D. R.; Gil, Y.; Joseph, R. L.; and Carbonell, J. G. 1989. PRODIGY 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University.

Minton, S.; Bresina, J.; and Drummond, M. 1991. Commitment strategies in planning: A comparative analysis. In *Proceedings of IJCAI-91*, 259–265.

Stone, P.; Veloso, M.; and Blythe, J. 1994. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems.*

Veloso, M. M. 1989. Nonlinear problem solving using intelligent casual-commitment. Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University.