Evaluating Nonlinear Planning

Stephen Soderland
Daniel S. Weld

Technical Report 91-02-03

Department of Computer Science and Engineering, FR-35
University of Washington, Seattle, WA 98195 USA

Evaluating Nonlinear Planning

Stephen Soderland
Daniel S. Weld
Department of Computer Science and Engineering[1]
University of Washington,
Seattle, WA 98105

January 29, 1991

Abstract

Although most people believe that nonlinear planning is more efficient than the linear approach, this intuition has never been formally justified nor empirically validated. In this paper we do both, characterizing the type of domains that offer the maximum performance differentiation and the features that distinguish the relative overhead of the two algorithms. As we expected, the nonlinear planner has a large advantage in temporally constrained domains where the specific order of plan steps is critical. Contrary to expectations, however, the linear algorithm had higher overhead for all but the smallest problems.

[1]This research was sparked by discussions with David McAllester who developed the algorithms, and benefited from conversations with Steve Hanks. Tony Barrett implemented the molecular biology domain and suggested the link between nonlinear heuristics and abstraction. This research was funded in part by National Science Foundation Grants IRI-8902010 and IRI-8957302, Office of Naval Research Grant 90-J-1904, and a grant from the Xerox corporation.

1 Introduction

Since the early work on NOAH [Sacerdoti, 1975], the common wisdom of the planning community has held that nonlinear planners, i.e. those that represent plans as partially ordered sets of actions, are more efficient than planners that quickly commit to a totally ordered sequence. It is somewhat surprising, therefore, that this intuition has never been formally justified.

In this paper we evaluate the efficiency of nonlinear planning both analytically and through empirical studies of randomly generated problems. We consider two main questions:

- What characteristics of a problem domain (if any) give an advantage to a nonlinear planning algorithm?
- How is the performance of linear and nonlinear planners affected by the amount of domain theory information?

The answer to the first question is not terribly surprising. The two planning approaches are comparable in performance when there are few constraints on the order of plan steps. However, the nonlinear approach is vastly superior when the domain is tightly constrained. Section 3 defends these claims both analytically and experimentally.

The answer to the second question is more surprising. Given the simplicity of the linear planner, we expected it to have lower overhead. However, this was not the case. Section 4 explains why the nonlinear approach is less affected by the number of initial conditions.

Although our analysis and experiments are specific to planners using the STRIPS representation, section 5 suggests possible extensions. In addition, we discuss other factors (amenability to heuristics and abstraction) that distinguish between the planning approaches.

2 Algorithms

Before presenting our results, we summarize the algorithms and representations implemented. Both linear and nonlinear planners use what is known as the STRIPS action representation [Chapman, 1987] although it is in fact a simplification of that used by STRIPS [Fikes and Nilsson, 1971, Davis, 1990]. Each operator has a set of preconditions, an add-list and a delete-list (the members of which are propositional schemata that are function-free atomic)

and a set of constraints. For example, the blocks world operator (puton ?x ?y) which takes block ?x from ?z and puts it on ?y has three preconditions: (on ?x ?z), (clear ?x), and (clear ?y). The operator adds (on ?x ?y) and (clear ?z), but deletes (on ?x ?z) and (clear ?y). The constraints specify that ?x, ?y, and ?z cannot be co-referential. Also neither ?x nor ?y can co-refer with table. The limitations of this action representation have been clearly documented in [Chapman, 1987]; never-the-less, we have succeeded in encoding a number of domains, including the blocks world, two artificial worlds, a discrete time version of Minton's scheduling world [Minton, 1988], and an approximation of Stefik's MOLGEN molecular biology domain [Stefik, 1981].

Both planners implemented were based on algorithms developed by David McAllester [McAllester, 1989, McAllester and Rosenblitt, 1991]. They each require three arguments: a set of domain operators, a set on initial conditions, and a set of goal conjuncts; they return sequences of steps. Both planners use least-commitment, constraint-posting techniques when reasoning about the arguments to the operators, and both planners operate via backward chaining. To ensure fairness, both planners were implemented in COMMON LISP using a shared set of data structures and subroutines. The most important such subroutine is the variable binding and unification code which handles both codesignation and noncodesignation constraints.[2]

2.1 The Nonlinear, Causal-Link Planner

The causal-link algorithm, developed by McAllester, is loosely descended from TWEAK [Chapman, 1987], but is conceptually simpler and has what McAllester terms the "systematic property" [McAllester and Rosenblitt, 1991]. Loosely speaking, this means that the planner is guaranteed to search the space of partial plans in an orderly fashion — visiting every possible plan exactly once. A stronger criterion than completeness, systematicity is an essential property used in our analysis of section 3.

McAllester's key innovation is the use of *causal links* that record why a step was introduced into a plan. If P is a proposition that is added by step S_i and is a precondition of step S_j, then $S_i \xrightarrow{P} S_j$ denotes the causal link. A link $S_i \xrightarrow{P} S_j$ is said to be *threatened* if a step S_k may possibly be ordered between S_i and S_j, and S_k deletes a proposition that possibly codesignates with P.

[2]To encourage experimentation, our planning and testing code is freely available for research purposes; contact one of the authors.

A nonlinear plan step S has an *open condition* if one of the preconditions of S has no causal link to an establishing step. The following algorithm builds a nonlinear plan by eliminating open conditions while ensuring the safety of threatened links.

Algorithm: Nonlinear, Causal-Link Planner (OPS, INITS, GOALS)

1. Initialize PRIORITY-QUEUE so it contains a single nonlinear plan with two ordered steps and no bindings. The initial step adds the propositions in INITS, but has neither preconditions nor a delete-list. The final step has GOALS as its preconditions, but has empty add and delete-lists.

2. If PRIORITY-QUEUE is empty, return *failure*, otherwise remove a nonlinear plan, PLAN, from PRIORITY-QUEUE.

3. If PLAN has no open preconditions and no threatened causal links then return a topological sort of the steps of PLAN after substituting for variables using bindings.

4. If some causal link $S_i \xrightarrow{P} S_j$ of PLAN is threatened, then select some step S_k of PLAN such that S_k deletes P and S_k might be between S_i and S_j and do the following:

   (a) If S_k might precede S_i, insert PLAN_1 to PRIORITY-QUEUE where PLAN_1 is derived by adding the ordering constraint $S_k < S_i$ to the partial ordering of PLAN.

   (b) If S_k might follow S_j, insert PLAN_2 to PRIORITY-QUEUE where PLAN_2 is derived by adding the ordering constraint $S_k > S_j$ to the partial ordering of PLAN.

   (c) For each consistent set of constraints that prevents S_k from deleting P, insert PLAN_3 to PRIORITY-QUEUE where PLAN_3 is derived from PLAN by adding the constraints to its bindings.

   (d) Go to step 2.

5. If no causal link in PLAN is threatened then select some open precondition P of a step S_j of PLAN and do the following:

   (a) For each operation O of OPS that possibly adds P, insert PLAN_4 to PRIORITY-QUEUE where PLAN_4 is derived from PLAN by adding a new step S_n of type O, by adding the causal link $S_n \xrightarrow{P} S_j$, by

constraining $S_n < S_j$, and by adding bindings that force S_n to add P.

(b) For each step S_i in PLAN that possibly adds P and that is possibly prior to S_j, insert PLAN_5 to PRIORITY-QUEUE where PLAN_5 is derived from PLAN by adding the causal link $S_i \xrightarrow{P} S_j$, by constraining $S_i < S_j$, and by adding bindings that force S_i to add P.

(c) Go to step 2.

A version of this algorithm has been proven sound, complete, and systematic [McAllester and Rosenblitt, 1991]. Note that if an appropriate ranking function is used for PRIORITY-QUEUE, then the algorithm performs A* search. We tested the algorithm with numerous search strategies. See section 5 for the relationship between various ranking functions and abstraction planning [Sacerdoti, 1974, Yang and Tenenberg, 1990].

2.2 The Linear Planner

In the treatment of [McAllester and Rosenblitt, 1991], the preceding algorithm is derived from a backward-chaining linear planner via what McAllester terms *quotient acceleration.* For our purposes, it suffices to note that the linear algorithm is the same as the nonlinear except it considers sequences of steps rather than partially ordered sets. Because of this simplification, the linear algorithm does not require data structures for links, ordering or open conditions, but it shares the data structures and routines for variable bindings and constraints. The algorithm works by building a plan backwards, considering all steps that could possibly achieve a subgoal without deleting any other subgoal; when it finds such a step it creates a new plan by eliminating the resolved subgoal(s) and adding the weakest preconditions for the new step as new subgoals. The planner terminates when all of the subgoals of a plan unify with the initial conditions.

Algorithm: Linear Planner (OPS, INITS, GOALS)

1. Initialize PRIORITY-QUEUE to contain the single partial plan with subgoals GOALS, no steps, and no bindings.

2. If PRIORITY-QUEUE is empty, return *failure*, otherwise remove a partial plan, PLAN, from PRIORITY-QUEUE.

3. If the subgoals of PLAN unify with a subset of the initial conditions yielding a substitution S that is consistent with the bindings of PLAN, then return the sequence of steps after substituting for variables using the bindings of PLAN.

4. For each operation O in OPS and each subset G of the subgoals of PLAN that is possibly added by O with bindings S that are consistent with the bindings of PLAN, insert PLAN_1 to PRIORITY-QUEUE where PLAN_1 is defined as follows.

   - The steps of PLAN_1 consist of an O-type step followed by the steps of PLAN.
   - The subgoals of PLAN_1 consist of the the subgoals of PLAN minus G union the preconditions of the new O step.
   - The bindings of PLAN_1 consist of the bindings of PLAN union S union whatever noncodesignation constraints are required to ensure that the delete-list of the new step does not even possibly interfere with a subgoal of PLAN. If multiple, minimal noncodesignation constraints fulfill this condition, then a new plan must be created for each such set.

5. Go to step 2.

Since step 4 of the linear algorithm has the potential to insert a large number of partial plans to the queue, we explored with a variety of search strategies, including iterative deepening, before settling on a lazy evaluation scheme in which successor plans are added gradually and in the same order by step 4 as they are in the nonlinear algorithm's step 5. This drastically decreased the linear algorithm's branching factor and sped the algorithm considerably in almost every test.

3 Effect of Flexibility in Plan Ordering

Common intuition suggests that a nonlinear planner has the biggest advantage over a linear planner when the steps to achieve a goal must be performed in a specific order. When there are no constraints on the ordering of steps in a plan, both linear and nonlinear planners should perform similarly, although with differing amounts of overhead. Although many people accept this characterization, it has never been justified formally. In this section we provide both

an analytic and empirical explanation of this phenomena. Then in section 4 we discuss the relative overhead of the two algorithms.

3.1 Unconstrained Ordering

3.1.1 Analytic Predictions

One can consider the search space of a nonlinear planner to be an abstraction of that of a linear planner. Let each node in the graph be a partial plan and each directed edge be labeled with the new step added to the plan. If various bindings are generated that allow the new step to resolve more than one goal, each combination forms a separate node in the search space. Likewise different nodes are created for each set of bindings that prevent the step from deleting subgoals. Thus a node may have several outgoing edges labeled with the same step.

Where the nonlinear has a single, unordered path labeled with the steps $\{A, B, C\}$, the linear planner may generate paths labeled $\{A<B<C\}$, $\{A<C<B\}$, $\{B<A<C\}$, $\{B<C<A\}$, $\{C<A<B\}$, and $\{C<B<A\}$ if the problem has no constraints eliminating any of these orderings. In general if a path in the nonlinear planner has n steps, there will be $n!$ corresponding paths in the linear search space.

One way to look at the effectiveness of the two algorithms is to compare their ratios of successful paths in the search space to the total number of paths. Consider the search tree to a depth of n. The success ratio for the nonlinear planner is

$$s/p$$

where s is the number of successful paths and p is the total number of paths. The success ratio for the linear planner is

$$\frac{sn!}{pn!} = s/p$$

since each path in the nonlinear corresponds to $n!$ paths in the linear search space. Thus nonlinear and linear planners should perform comparably in the case where the order of steps doesn't matter.

3.1.2 Empirical Results

To test this prediction, we created an artificial domain theory, called ART1, that has fifteen possible action types that each achieve a different predicate

when their individual initial condition is present. Each action type is independent — they overlap neither preconditions nor add lists, and every action's delete list is empty. As a result, the order in which a plan's subgoals are handled, and the eventual ordering of the steps was predicted to be irrelevant to either algorithm.

Given this domain, we generated 75 solvable problems each consisting of 15 randomly permuted initial conditions and between 1 and 15 randomly selected and permuted goal propositions such that 5 problems were generated for each number of goal conjunctions. Each problem was given to both algorithms; the results are shown in figure 1. Each point on the graph represents the average of five random tests with that number of conjuncts. Performance was measured in milliseconds of Dec 5000 CPU time.

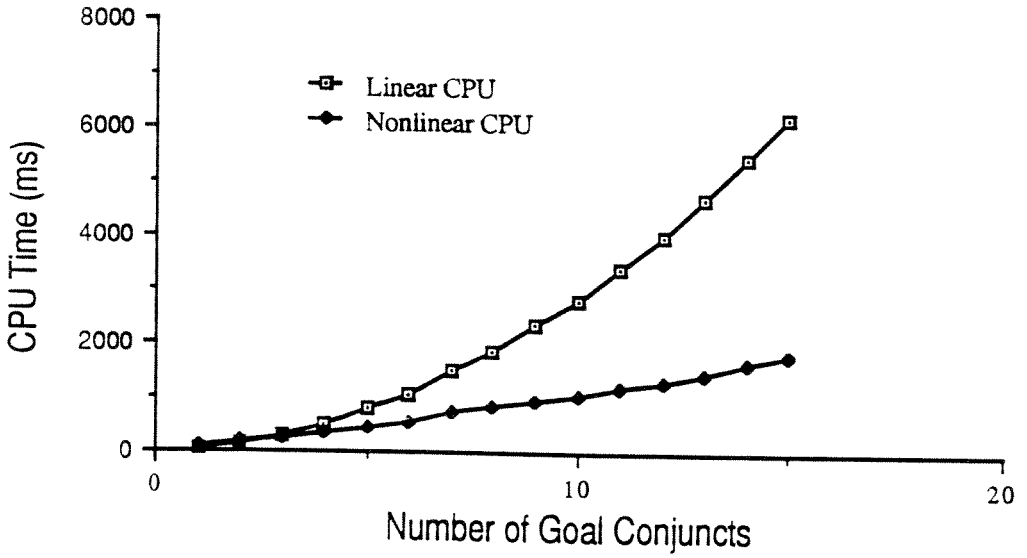

Figure 1: Both Planners can Handle the Unconstrained Domain

It is clear from the graph that both planning algorithms have close to linear time complexity in the number of goals for this unconstrained domain. Graphs showing the number of partial plans created during the search have an identical shape. It must be noted that the performance shown is dependent on the fact that only solvable problems were generated. While the nonlinear algorithm would have quickly quit attempting an impossible goal in this domain, the linear algorithm would have explored an exponential number of plans in a futile attempt to find a satisfactory ordering. Similarly, the performance of the linear planner is dependent on the use of an aggressive search strategy (i.e.

depth-oriented search), while the nonlinear planner's smaller overall search space renders it less dependent on a particular strategy.

The reader may be surprised that the simple linear planner appears twice as slow as its nonlinear sibling — even in a domain carefully constructed to neutralize the nonlinear advantage. This is explained in section 4.

3.2 Constrained Ordering

Now we consider the case where nonlinear planners should prove their worth, the case where step-ordering is highly constrained.

3.2.1 Analytic Predictions

In the extreme case there is only one possible ordering of any pair of steps. If the linear planner generates a plan with steps $\{A{<}B{<}C{<}D\}$, no other order of those steps is possible. The plan $\{A{<}B{<}D\}$ or $\{A{<}C\}$ could be generated, since the steps included are in order, but $\{A{<}D{<}C{<}B\}$ will never be a path in the search space.

For each path in the nonlinear search space with n steps, there will now be 2^{n-1} paths in the linear search space. This exponential growth can be shown inductively, since increasing the length of the nonlinear plan from n to $n+1$ doubles the number of paths. If the path begins with the correct step it continues with the possible paths for the remaining n steps. Otherwise it can never include that step without violating ordering constraints and continues with the possible paths for the other n steps.

The success ratio for the nonlinear planner is the same as before, s/p, where s is the number of successful paths and p is the total number of paths. But now each successful nonlinear path corresponds to 2^{n-1} paths in the linear search space to a depth of n. This gives

$$\frac{s}{p2^{n-1}}$$

which is worse than the nonlinear success ratio by an exponential factor.

3.2.2 Empirical Results

To test this prediction, we created an artificial domain theory, called ART2, which resembles ART1 except that the temporal ordering of plan steps is tightly constrained by the delete lists of each action. For any set of goals, there exists a single ordering of steps that will achieve the goal; switching the

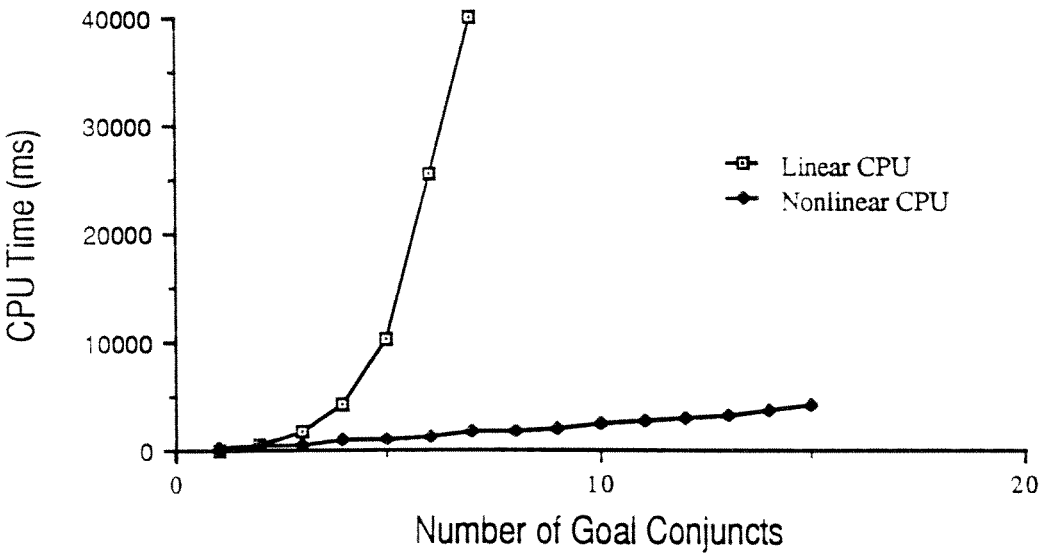

Figure 2: Nonlinear Planner Excels in Constrained Domains

i + 1st step before the ith causes the i + 1st to delete the preconditions for the ith. In all other respects, the action types are independent and identical to those of ART1.

Given this domain, we generated 75 solvable problems in the same fashion as the previous experiment: 5 random problems for each number of goal conjuncts between 1 and 15. As shown in figure 2, the nonlinear planner maintained its near linear time complexity, while the linear planner was incapable of solving even moderately sized problems before the resource cutoff. A graph of the number of partial plans created had an identical shape. The performance of the linear planner was not improved by changing the search strategy. The conclusion is clear — domains with tight interaction between the ordering of steps from different subgoals provide an extreme advantage for the nonlinear algorithm.

4 Effect of Domain Theory Size

The linear and nonlinear algorithms share many of the costs of generating new plans. A fair way to estimate these costs is by the number of unifications requested since this correlates well ($R^2 \geq 0.98$) with CPU time. By this metric, the cost of finding a step to establish each goal is the same in the two algorithms, and the cost of testing for possible conflicts from deletes is comparable. The nonlinear planner's adding of new causal links has basically the same cost as the linear planner finding all bindings for a new step that

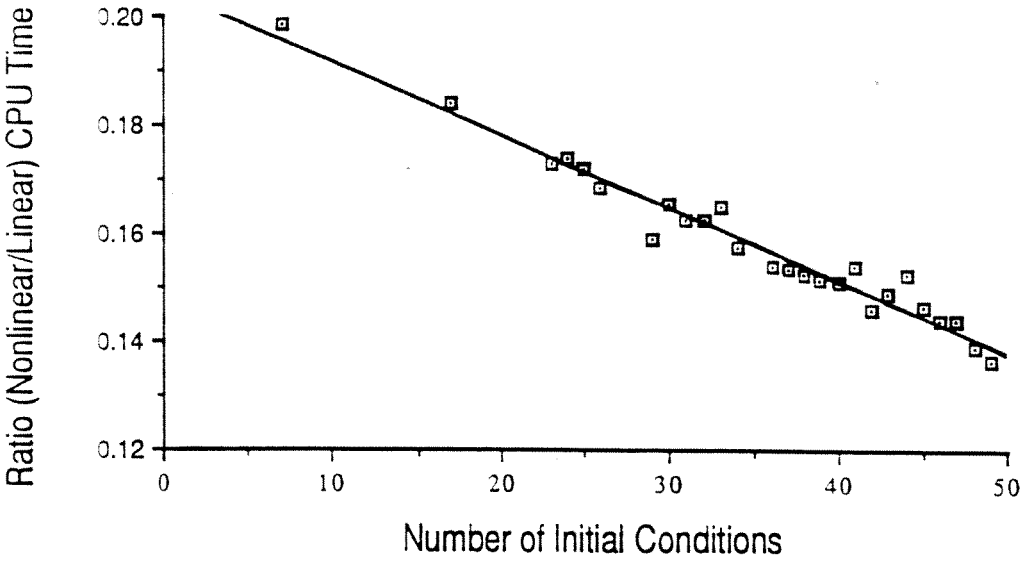

Figure 3: Nonlinear Planner Degrades More Slowly

resolve additional subgoals.

The biggest difference between the per-plan cost of the two approaches concerns matches to the initial conditions. While the nonlinear algorithm has a list of open goals that keeps getting smaller until the plan is completed, the linear planner has no way to remove goals that unify with the initial conditions. The only way for the linear algorithm to know if a plan is complete is to see if its subgoals unify with a subset of the initial conditions, which can be an expensive test. In practice, this must be done every time a new plan is created, since any reasonable heuristic will take into account the number of open goals.

The nonlinear algorithm escapes this penalty by systematically considering which subgoals could be established by the initial conditions and using causal links to record this information. Since this decision is made on a per goal basis not a per plan basis, the overhead is reduced. From this analysis, we predicted that the linear planner's overhead should rise more quickly than that of the nonlinear algorithm as the number of initial conditions increased.

To test this conjecture, we generated 300 blocks world problems by generating a random problem and then adding varying amounts of random irrelevant initial conditions. Figure 3 shows the results of the experiment — each point represents the ration of nonlinear to linear CPU time averaged over all problems with that number of initial conditions. As is apparent from the best-fitting line (correlation $R^2 = 0.96$), the nonlinear algorithm takes proportionally less time as the number of initial conditions grows.

5 Discussion

In order for the field of artificial intelligence planning to mature, it is necessary to precisely characterize the behavior of central algorithms. We view this paper as the logical successor to Korf's delineation of the effect of decomposition and abstraction in planning [Korf, 1987].

This paper used both analytic arguments and empirical evidence to evaluate nonlinear planning. We showed that while linear and nonlinear planners perform similarly in domains where step order is unrestricted, the nonlinear algorithm dominates in cases where the order of plan steps is tightly constrained. In addition, we showed that the relative performance of the nonlinear planner improves as the number of initial conditions increases.

With any empirical comparison, there is a possible question of fairness. By writing the planners in the same language, using the same data structures the same subroutines, and the same basic algorithm (modulo quotienting [McAllester and Rosenblitt, 1991]), we feel that fairness is not a concern.

While our empirical results and the analysis of section 4 are are specific to backward chaining planners, we suspect that they could be extended without much difficulty. A more serious restriction is the reliance of our evaluation on STRIPS representation. The lack of functions and conditionals is directly exploited by the causal-link algorithm as it was earlier by TWEAK. With this representation, the equivalent of an exponential number of linear plans can be checked in polynomial time. Unfortunately, the work of Dean and Boddy [Dean and Boddy, 1988] suggests that the domination of linear by nonlinear planning may not extend to more expressive action representations. This is an open question that deserves more attention.

One advantage of nonlinear planning that may extend to other action representations has not been treated quantitatively in this paper. A nonlinear planner is more amenable to advice than a linear algorithm; the algorithms can exploit different types of heuristics. For example, the ideal, linear, goal-ordering heuristic must dictate which subgoal needs to be tackled next in the time-frame of the executor, while the nonlinear equivalent must only say which subgoal is the most critical to achieve in the context of the planning process. Another way to view this is to realize that for the causal-link algorithm, a goal-ordering heuristic that considers only the predicate type, is equivalent to an abstraction planner based on predicate criticalities [Sacerdoti, 1974, Yang and Tenenberg, 1990]. This connection to abstraction supports our conjecture that for most domains it is easier to develop heuristics for a nonlinear planner.

References

[Chapman, 1987] D. Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32(3):333–377, July 1987.

[Davis, 1990] E. Davis. *Representations of Commonsense Knowledge.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

[Dean and Boddy, 1988] T. Dean and M. Boddy. Reasoning about Partially Ordered Events. *Artificial Intelligence*, 36(3), October 1988.

[Fikes and Nilsson, 1971] R. Fikes and N. Nilsson. STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3/4), 1971.

[Korf, 1987] R. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, 33(1), September 1987.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Submitted to AAAI-91*, July 1991.

[McAllester, 1989] D. McAllester. Course Notes for MIT 6.824 Artificial Intelligence. Unpublished, MIT AI Lab, Fall 1989.

[Minton, 1988] S. Minton. Quantitative Results Concerning the Utility of Explanation-Based Learning. In *Proceedings of AAAI-88*, pages 564–569, August 1988.

[Sacerdoti, 1974] E. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Sacerdoti, 1975] E. Sacerdoti. The Nonlinear Nature of Plans. In *Proceedings of IJCAI-75*, pages 206–214, 1975.

[Stefik, 1981] M. Stefik. Planning with Constraints (MOLGEN: Part 1). *Artificial Intelligence*, 14(2), 1981.

[Yang and Tenenberg, 1990] Q. Yang and J. Tenenberg. ABTWEAK: Abstracting a Nonlinear, Least-Commitment Planner. In *Proceedings of AAAI-90*, pages 204–209, August 1990.