Evaluating the Trade-offs in Partial-Order Planning Algorithms

Craig A. Knoblock
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

Qiang Yang
University of Waterloo
Computer Science Department
Waterloo, Ont., Canada N2L 3G1

May 1994
ISI/RR-94-381
Evaluating the Trade-offs in Partial-Order Planning Algorithms

Craig A. Knoblock
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

Qiang Yang
University of Waterloo
Computer Science Department
Waterloo, Ont., Canada N2L 3G1

May 1994
ISI/RR-94-381

In proceedings of the Canadian Artificial Intelligence Conference
Banff, Alberta, Canada, May 1994
Evaluating the Trade-offs in Partial-Order Planning Algorithms

Craig Knoblock
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292

Qiang Yang
University of Waterloo

Through a comparison of the three algorithms, we provide a detailed analysis of different goal protection methods, in order to identify the facts that determine the performance of the systems. The analysis clearly shows that the relative performance of the different goal-protection methods used by the systems, depends on the characteristics of the problems being solved. One of the main determining factors of performance is the ratio of the number of negative threats to the number of positive threats. We present an artificial domain where we can control this ratio and show that in fact the planners show radically different performance as the ratio is varied. The implication of this result for someone implementing a planning system is that the most appropriate algorithm will depend on the types of problems to be solved by the planner.
**INSTRUCTIONS FOR COMPLETING SF 298**

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

<table>
<thead>
<tr>
<th>Block 1. Agency Use Only (Leave blank).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.</td>
</tr>
<tr>
<td>Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).</td>
</tr>
<tr>
<td>Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When the report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.</td>
</tr>
<tr>
<td>Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:</td>
</tr>
<tr>
<td>C - Contract</td>
</tr>
<tr>
<td>G - Grant</td>
</tr>
<tr>
<td>PE - Program</td>
</tr>
<tr>
<td>Element</td>
</tr>
<tr>
<td>Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).</td>
</tr>
<tr>
<td>Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.</td>
</tr>
<tr>
<td>Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.</td>
</tr>
<tr>
<td>Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory.</td>
</tr>
<tr>
<td>Block 10. Sponsoring/Monitoring Agency Report Number. (If known)</td>
</tr>
<tr>
<td>Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with..., Trans. of ..., To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.</td>
</tr>
<tr>
<td>Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).</td>
</tr>
<tr>
<td>DOD - See DoDD 5230.24, &quot;Distribution Statements on Technical Documents.&quot;</td>
</tr>
<tr>
<td>DOE - See authorities.</td>
</tr>
<tr>
<td>NTIS - Leave blank.</td>
</tr>
<tr>
<td>Block 12b. Distribution Code.</td>
</tr>
<tr>
<td>DOD - Leave blank.</td>
</tr>
<tr>
<td>DCE - Enter DOE distribution categories from the Standard Distribution for Classified Scientific and Technical Reports.</td>
</tr>
<tr>
<td>NASA - Leave blank.</td>
</tr>
<tr>
<td>NTIS - Leave blank.</td>
</tr>
<tr>
<td>Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.</td>
</tr>
<tr>
<td>Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.</td>
</tr>
<tr>
<td>Block 15. Number of Pages. Enter the total number of pages.</td>
</tr>
<tr>
<td>Block 16. Price Code. Enter appropriate price code (NTIS only).</td>
</tr>
<tr>
<td>Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.</td>
</tr>
</tbody>
</table>

Standard Form 298 Back (Rev. 2-89)
Evaluating the Tradeoffs in Partial-Order Planning Algorithms

Craig A. Knoblock
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
knoblock@isi.edu

Qiang Yang
University of Waterloo
Computer Science Department
Waterloo, Ont., Canada N2L 3G1
qyang@logos.uwaterloo.ca

Abstract
Most practical partial-order planning systems employ some form of goal protection. However, it is not clear from previous work what the tradeoffs are between the different goal-protection strategies. Is it better to protect against all threats to a subgoal, some threats, or no threats at all? In this paper, we consider three well-known planning algorithms, SNLP, NONLIN, and TWEAK. Each algorithm makes use of a different goal-protection strategy. Through a comparison of the three algorithms, we provide a detailed analysis of different goal protection methods, in order to identify the factors that determine the performance of the systems. The analysis clearly shows that the relative performance of the different goal-protection methods used by the systems, depends on the characteristics of the problems being solved. One of the main determining factors of performance is the ratio of the number of negative threats to the number of positive threats. We present an artificial domain where we can control this ratio and show that in fact the planners show radically different performance as the ratio is varied. The implication of this result for someone implementing a planning system is that the most appropriate algorithm will depend on the types of problems to be solved by the planner.

1 Introduction
There has been a great deal of work recently on comparing total and partial order planning systems [Barrett

The first author is supported by Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency under contract no. F30602-91-C-0081. The second author is supported in part by grants from the Natural Science and Engineering Research Council of Canada, and ITRC: Information Technology Research Centre of Ontario. The views and conclusions contained in this paper are the author’s and should not be interpreted as representing the official opinion or policy of DARPA, RL, NSERC, ITRC, or any person or agency connected with them.

and Weld, 1992; Minton et al. 1991], but little has been done in comparing different partial order planners themselves. There are a variety of design decisions that must be made in order to build a general planner. This paper focuses on one of these design choices – the choice of a protection strategy. In particular, we compare the protection strategy employed in three basic planning algorithms, SNLP, NONLIN and TWEAK.

On the surface, the three planners are quite different. However, on a careful examination one can find they mainly differ in which conditions they protect. During planning, an inserted plan step can interact with previously inserted steps. If a goal is achieved by one plan step, then later it could be threatened by other steps. A goal is protected by removing all threats by imposing additional constraints on a plan whenever a threat is detected. Among the three planners, TWEAK protects nothing, NONLIN protects against all negative threats, and SNLP protects against both negative and positive threats.

The use of goal protection in SNLP prevents the planner from generating redundant plans and thereby could potentially reduce the size of the search space. However, enforcing the goal protection has a cost. In this paper, we show that none of the planners is always a winner. In some domains our planner based on TWEAK greatly outperforms both a planner based on NONLIN 1 and SNLP. In other domains, SNLP and NONLIN perform much better than TWEAK. The challenge is to identify the features of the domains where each planner is expected to perform well, so that practitioners can balance the protection methods based on the application domain.

In the following sections, we first review the three algorithms. Then we present an analysis of the algorithms to identify their relative merits. We also report on two critical domain features that have the greatest impact on the performance of the planners. Finally, we present empirical results on an artificial domain to support the analysis.

1 For convenience we will simply refer to them as TWEAK and NONLIN.
2 Comparison of the Algorithms

This section presents the SNLP, TWEAK, and NONLIN planning algorithms. First, we present the SNLP algorithm based on the algorithm descriptions of McAllester and Rosenblitt's Find-Completion algorithm [McAllester and Rosenblitt, 1991] and Barrett and Weld's POCL algorithm [Barrett and Weld, 1992]. We start with this algorithm because we can build on the elegant algorithm description and implementation provided in previous work. Then, we describe the changes necessary to transform the SNLP algorithm into algorithms that implement NONLIN [Tate, 1977] and TWEAK [Chapman 1987].

2.1 The SNLP Algorithm

In the planning algorithms that we consider below, we follow the notations used by Barrett and Weld [Barrett and Weld, 1992]. A plan is a 3-tuple, represented as \( \langle S, O, B \rangle \), where \( S \) is a number of steps, \( O \) is a set of ordering constraints, and \( B \) the set of variable binding constraints associated with a plan. A step consists of a set of preconditions, an add list, and a delete list. The binding constraints specify whether two variables can be bound to the same constant or not.

The core of SNLP is the recording of the causal links for why a step is introduced into a plan, and for protecting that purpose. If a step \( S_i \) adds a proposition \( p \) to satisfy a precondition of step \( S_j \), then \( S_i \rightarrow S_j \) denotes the causal link. An operator \( S_k \) is a threat to \( S_i \rightarrow S_j \) if \( S_k \) can possibly add or delete a literal \( q \) that can possibly be bound to \( p \). For convenience, we also refer to the pair \( (S_k, S_i \rightarrow S_j) \) as a threat. In addition, we define an operator \( S_k \) to be a positive threat to \( S_i \rightarrow S_j \) if \( S_k \) can possibly be between \( S_i \) and \( S_j \), and \( S_k \) adds a literal \( q \) that can possibly be bound to \( p \). Likewise, \( S_k \) is a negative threat if it can possibly be between \( S_i \) and \( S_j \), and deletes a literal \( q \) that can possibly be bound to \( p \).

The following algorithm which is an adaptation of McAllester and Rosenblitt's Find-Completion algorithm [McAllester and Rosenblitt, 1991] and Barrett and Weld's POCL algorithm [Barrett and Weld, 1992], has been shown to be sound, complete, and systematic (never generates redundant plans). Let the notation codesignate\((R)\) denote the codesignation constraints imposed on a set of variable pairs \( R \). For example, if \( R = \{(x_i, y_i) \mid i = 1, 2, \ldots, k \} \), then codesignate\((R) = \{x_i = y_i \mid i = 1, 2, \ldots, k\} \). Similarly, noncodesignate\((R)\) denotes the set of non-codesignation constraints on a set \( R \) of variable pairs. The parameters of the algorithm are: \( S = \text{Steps}, O = \text{Ordering constraints}, B = \text{Binding constraints}, G = \text{Goals}, T = \text{Threats}, \) and \( L = \text{Causal links} \).

Algorithm SNLP\((\langle S, O, B \rangle, T, G, L)\)

1. Termination: If \( G \) and \( T \) are empty, report success and stop.
2. Declobbering: A step \( s_k \) threatens a causal link \( s_i \rightarrow s_j \) when it occurs between \( s_i \) and \( s_j \), and it adds or deletes \( p \). If there exists a threat \( t \in T \) such that \( t \) is a threat between a step \( s_k \) and a causal link \( s_i \rightarrow s_j \), then:
   - Remove the threat by adding ordering constraints and/or binding constraints using promotion, demotion, or separation. For completeness, all ways of resolving the threat must be considered.
   - Promotion: \( O' = O \cup \{s_k \rightarrow s_j\}, B' = B \)
   - Demotion: \( O' = O \cup \{s_j \rightarrow s_k\}, B' = B \)
   - Separation: \( O' = O \cup \{s_k \rightarrow s_j\} \cup \{s_k \rightarrow s_j\} \). Let \( q \) be the effect of \( s_k \) that threatens \( p \) and let \( P \) be the set of binding pairs between \( q \) and \( p \). \( B' = B \cup \sigma \), where \( \sigma = \{ \alpha \mid \alpha = \text{noncodesignate}(s) \}\cup \text{codesignate}(P - s) \), where \( s \subseteq P \land s \neq \emptyset \).
   - Recursive invocation: \( \text{SNLP}(\langle S, O', B' \rangle, T - \{t\}, G, L) \)

3. Goal selection: Let \( p \) be a proposition in \( G \), and let \( S_{\text{need}} \) be the step for which \( p \) is a precondition.
4. Operator selection: Let \( S_{\text{add}} \) be an existing step, or some new step, that adds \( p \) before \( S_{\text{need}} \). If no such step exists or can be added then backtrack. Let \( L' = L \cup \{(s_{\text{add}}, S_{\text{need}})\} \), \( S' = S \cup \{S_{\text{add}}\} \), \( O' = O \cup \{s_{\text{add}} \rightarrow S_{\text{need}}\} \), and \( B' = B \cup \) the set of variable bindings to make \( S_{\text{add}} \) add \( p \). Finally, update the goal set: \( G' = (G - \{p\}) \cup \) preconditions of \( S_{\text{add}} \), if new. For completeness, all ways of achieving the step must be considered.
5. Threat identification: Let \( T' = \{t \mid \text{for every step } s_k \text{ that is a positive or negative threat to a causal link } s_i \rightarrow s_j \in L', t = (s_k, s_i \rightarrow s_j)\} \).
6. Recursive invocation: \( \text{SNLP}(\langle S', O', B' \rangle, T', G', L') \).

2.2 The NONLIN Algorithm

SNLP is a descendant of NONLIN [Tate, 1977], so the algorithms are quite similar and differ mainly in which threats they protect against and how they perform separation. These two differences stem from the added constraints on SNLP that are used to ensure systematicity. NONLIN also provides some additional capabilities such as hierarchical task-network decomposition, but these capabilities are orthogonal to the point of this paper and are not considered.

The first change to the SNLP algorithm is in the threat identification step. In contrast to SNLP, only the negative threats are added to the list \( T' \):

Threat identification: Let \( T' = \{t \mid \text{for every step } s_k \text{ that is a negative threat to a causal link } s_i \rightarrow s_j \in L', t = (s_k, s_i \rightarrow s_j)\} \).

The second change is that to perform separation, there is no requirement that promotion, demotion and separation are made mutually exclusive. In this case, separation simply entails that one or more of the possible bindings are forced not to codesignate, but imposes no ordering constraints.

Increasing completeness, all ways of resolving the threat must be considered.
Separation: $O' = O$. Let $q$ be the effect of $e_b$ that possibly codesignates with $p$ and let $P$ be the set of binding pairs between $q$ and $p$. $B' = B \cup \sigma$, where $\sigma \in \alpha \setminus \alpha = \text{noncodesignate}(e)$, where $e \in P$.

As we will see in the experimental results section, the differences in performance of goal protection methods employed by SNLP and NOLIN are relatively minor.

### 2.3 The TWEAK Algorithm

The primary difference between TWEAK and the two previous algorithms is that instead of building explicit causal links for each precondition established by the planner, TWEAK uses what is called the Modal Truth Criterion [Chapman 1987] to check the truth of each precondition in the plan. This difference results in four changes from the SNLP algorithm and only three changes from the NONLIN algorithm. The differences are in termination, separation, goal selection, and threat identification. Each of these is discussed in turn.

Since TWEAK does not maintain explicit causal links for each precondition, it must test the truth of all of the preconditions in the plan to determine when the plan is complete. It does this using the Modal Truth Criterion check [Chapman 1987]. This algorithm takes $O(n^3)$ time, as compared with the $O(1)$ time termination routine of SNLP. We will refer to the algorithm that implements the Modal Truth Criterion as $mtc$. This algorithm returns true if a given plan is complete and otherwise returns a precondition of some step in the plan that does not necessarily hold.

**Termination:** If $mtc((S, O, B))$ is true, report success and stop.

**Similar to NONLIN,** there is no requirement that all of the separation constraints are mutually exclusive. Thus, TWEAK uses the same method for separation as NONLIN.

**Separation:** $O' = O$. Let $q$ be the effect of $e_b$ that possibly codesignates with $p$ and let $P$ be the set of binding pairs between $q$ and $p$. $B' = B \cup \sigma$, where $\sigma \in \alpha \setminus \alpha = \text{noncodesignate}(e)$, where $e \in P$.

Since TWEAK does not maintain an explicit set of causal links, there is no explicit record of which preconditions must be achieved. Thus, goal section is done and separation, goal selection, and threat identification are no attempt to protect all of the previously established preconditions against either negative or positive threats. TWEAK does, however, ensure that at each step all negative threats to the most recently built causal link are removed. However, after a precondition is established and threats are removed, it can be clobbered again. In such a case, TWEAK will have to re-establish the condition.

**Threat identification:** Let $h_{new} = S_{\text{add}} \cap S_{\text{need}}$, which is the causal link constructed in step 4. Let $T = \{\}$ for every step $e_b$ that is a negative threat to $h_{new}$, $t = (s_h, h_{new})$.

As we stated above, the $mtc$ routine for the termination check is more expensive than that for SNLP. However, this does not mean that TWEAK is less efficient than SNLP, since in many cases, TWEAK will explore fewer nodes. In the next section, we consider the major factors that affect the search space, and present a complexity analysis of the three algorithms.

### 3 Analyzing the Algorithms

#### 3.1 Algorithm Complexities

Let $eb$ be the effective branching factor and $ed$ the effective depth of the search tree. In both algorithms, $eb$ is the maximum number of successor plans generated either after step 2, or after step 5, while $ed$ is the maximum number of plan expansions in the search tree from the initial plan state to the solution plan state. Then with a breadth-first search, the time complexity of search is

$$O(eb^ed * T_{\text{node}}),$$

where $T_{\text{node}}$ is the amount of time spent per node.

We next analyze the complexity of the algorithms by fleshing out the parameters $eb$, $ed$ and $T_{\text{node}}$. In this analysis, let $P$ denote the maximum number of preconditions or effects for a single step, let $N$ denote the total number of operators in an optimal solution plan, and let $A$ be either the SNLP, NONLIN, or TWEAK algorithm.

To expand the effective branching factor $eb$, we first define the following additional parameters. We use $b_{new}$ for the number of new operators found by step 4 for achieving $p$, $b_{old}$ for the number of existing operators found by step 4 for achieving $p$, and $e_{\text{new}}$ for the number of alternative constraints to remove one threat. The effective branching factor of search by either algorithm is then

$$eb = \max\{b_{new} + b_{old}, r_{\text{new}}\},$$

since each time the main routine is followed, either step 2 is executed for removing threats, or step 3-6 is executed to build causal links. If step 2 is executed, $r_{\text{new}}$ successor states are generated, but otherwise, $(b_{new} + b_{old})$ successor plan states are generated.

Next, we expand the effective depth $ed$. In the solution plan, there are $N * P$ number of $(p, S_{\text{need}})$ pairs, where $p$ is a precondition for step $S_{\text{need}}$. Let $f_A$ be the fraction of the $N * P$ pairs chosen by step 3. For each pair $(p, S_{\text{need}})$ chosen by step 3, step 5 accumulates a set of threats to remove. Let $t_A$ be the number of threats generated by step 5. Finally, $v$ be the total number of times any fixed pair $(p, S_{\text{need}})$ is chosen by step 3. Then we have

$$ed_A = f_A * N * P * t_A * v_A.$$
Table 1: Parameters used in complexity analysis.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eb</td>
<td>effective branching factor</td>
</tr>
<tr>
<td>ed</td>
<td>effective search depth</td>
</tr>
<tr>
<td>T_node</td>
<td>average time per node</td>
</tr>
<tr>
<td>N</td>
<td>total number of operators in a plan</td>
</tr>
<tr>
<td>P</td>
<td>total number of preconditions per operator</td>
</tr>
<tr>
<td>f_A</td>
<td>fraction of ((p, S_{\text{need}})) pairs examined by algorithm A</td>
</tr>
<tr>
<td>v_A</td>
<td>average number of times a ((p, S_{\text{need}})) pair is visited by A</td>
</tr>
<tr>
<td>t_A</td>
<td>average number of threats found by A at each node</td>
</tr>
<tr>
<td>r_A</td>
<td>average number of ways to resolve a threat by A</td>
</tr>
<tr>
<td>b_new</td>
<td>average number of new threat establers for a precondition</td>
</tr>
<tr>
<td>b_old</td>
<td>average number of existing (or old) threat establers for a precondition</td>
</tr>
</tbody>
</table>

4 Domain Features and Search Performance

The analysis in the previous section can be used to predict the relative performance of the three planning algorithms in different types of domains. An important feature of a domain that determines the relative performance of any two algorithms is the ratio between the number of positive threats and number of negative threats. The ratio is an important factor in differentiating the algorithms because the major difference between any two algorithms is the way they handle positive and negative threats. Among the three algorithms, TWEAK only avoids some negative threats, SLP protects against all positive and negative threats, and NONLIN protects against all negative threats but not the positive ones.

4.1 Predictions

The major difference between the algorithms manifest themselves in the execution of Step 1, the termination subroutine, and Step 4, threat detection. To see their effect on search efficiency, let \(t_+\) denote the average number of positive threats, and let \(t_-\) be the average number of negative threats detected by Step 4 of SLP. Let \(R\) denote the ratio of \(t_-\) to \(t_+\): \(R = \frac{t_-}{t_+}\). In this section we predict the performance of the three planning algorithms based on the value of \(R\).

Case 1: \(R \ll 1\)

Since SLP resolves all positive threats, it imposes more constraints on a plan. Thus, on the average an SLP plan is more linearly ordered than either a TWEAK plan or a NONLIN plan. A more linearly ordered plan has a smaller number of existing establishing operators for a given precondition, and thus a smaller branching factor. Thus, the branching factor of SLP is likely to be the smallest among the three, and that for TWEAK is the largest due to its conservative stand in resolving threats.

When \(t_-\) is relatively large, the total number of threats \(t\) resolved by SLP is large, which in turn increases SLP's search depth. Also, for both NONLIN and SLP, a causal link has to be built for every precondition in a plan, a behavior that fixes a lower bound on their search depths. With many positive threats in a plan, a
precondition is more likely to be achievable by an existing step. Therefore TWEAK will be able to skip many more preconditions compared to NONLIN and SNLP. Thus the search depth of TWEAK will be much less than both NONLIN and SNLP, and the search depth of NONLIN will be smaller than SNLP because it does not resolve positive threats.

As R decreases below one, the branching factor for TWEAK and NONLIN increase, while the search depth for SNLP increases. The time complexity for the former goes up polynomially, while for the latter it goes up exponentially. Moreover, the depth of NONLIN is greater than the depth of TWEAK. Therefore, we predict that when \( R < 1 \) TWEAK will perform better than NONLIN, which in turn will perform better than SNLP.

Case 2: \( R \approx 1 \)

As with the previous case, the additional constraints imposed by SNLP and NONLIN over TWEAK imply that SNLP will have a smaller branching factor than NONLIN, and NONLIN will have a smaller branching factor than TWEAK. However, the difference in the number of threats \( t \) resolved by TWEAK, SNLP, and NONLIN will be reduced since there are fewer positive threats and more negative threats. The reduced number of positive threats will reduce the depth for SNLP and NONLIN and the increased number of negative threats increases the chance that TWEAK will be forced to revisit the same precondition/step pair. As a result, the performance of the different planners could be very close and will depend on depth and branching factors for the problems being solved.

Case 3: \( R \gg 1 \)

TWEAK is likely to have the largest branching factor because every time a negative threat occurs, all existing and new operators are considered as establishers again. This effect increases the factor \( b.odd \) for TWEAK, resulting in the effective branching factor for TWEAK being greater than both SNLP and NONLIN. Also due to its resolution of positive threats, a SNLP plan is likely to be more linearized than a NONLIN plan, thus the branching factor of SNLP will be smaller than NONLIN.

Each negative threat creates a chance for TWEAK to revisit the same precondition/step pair. Since in the \( R \gg 1 \) case, there is a large number of negative threats, the number of times each precondition is visited, \( v_{TWEAK} \), is likely to increase. Since TWEAK is expected to have a larger branching factor and depth greater than both SNLP and NONLIN, when \( R \gg 1 \) TWEAK is expected to perform the worst. SNLP will outperform NONLIN slightly due its smaller branching factor.

4.2 Empirical Results

In order to verify our predictions by comparing SNLP, NONLIN and TWEAK on problems with different ratios of negative and positive threats, we constructed an artificial domain where we could control the value of \( R \). In this domain, each goal can be achieved by a subplan of two steps in a linear sequence. Each step either achieves a goal condition or a precondition of a later step. The preconditions of the first step always hold in the initial state. In addition, we also added extra operator effects to create threats in planning. The difficulty of the problems in this domain can be increased by increasing the number of goal conditions and the total number of threats.

We used this artificial domain to run a set of experiments to compare the performance of the different planners. In these experiments we simultaneously varied the number of positive and negative interactions, such that the total number of interactions remained the same, but the ratio \( R \) changes from zero to infinity; the number of negative interactions increased from 0 to 9 while the number of positive interactions decreased from 9 to 0. Below, we present the results of our empirical tests on different points of the spectrum of as defined by the ratio \( R \).

In the experiments, each problem was run in SNLP [Barrett and Weld, 1992], a version of NONLIN and a version of TWEAK that were modified from SNLP. The problems were solved using a best-first search on the solution size in order to fairly compare the size of the problem spaces being searched by each system. All the problems were run on a SUN IPC in Lucid Common Lisp with a 120 CPU second time bound. For each value of ratio \( R \), we ran the systems on 20 randomly generated problems. The points shown in the graphs below are an average of the 20 problems.

4.2.1 Branching Factor

The branching factor results are shown in Figure 1. Most of our predictions for branching factors are observable in the figure. For example, due to its conservative stand in resolving both positive and negative threats, SNLP imposes the most constraints onto a plan, and as a results it generally has the lowest branching factor. Also, as the number of negative threats increases, which constrains the possible plans, the branching factor decreases to one.

However, there are a few surprises shown in the figure. When \( R < 1 \), we had predicted that TWEAK would have a larger branching factor than SNLP and would be similar to NONLIN. This prediction cannot be observed from the figure. In order to explain this effect we have broken the branching factor into the two parts described in the analysis, the establishment branching factor and the decluttering branching factor, which are combined to form the overall branching factor. These graphs are shown in Figures 2 and 3. As shown in the graphs, the smaller than expected branching factor for TWEAK is due to a smaller than expected establishment branching factor.
Careful analysis of the data shows that this discrepancy with the predictions is due to the assumption that the branching factor is uniform across an entire problem-solving episode. In fact, where there are many positive interactions, TWEAK quickly narrows in on a plan and reduces the establishment branching factor. In contrast, because both SNLP and NONLIN build explicit causal links and resolve more threats they spend more time in the early plan formation stage when the branching factor is higher. Thus overall, SNLP and NONLIN expand a larger part of the search space that has a large branching factor, while TWEAK uses its ability to exploit positive threats to rapidly traverse that part of the search space.

4.2.2 Depth

The comparison of the search depths is shown in Figure 4 and they are as predicted. The only apparent discrepancy is that the difference between SNLP and TWEAK should be larger when $R \ll 1$. However, the graph is a bit misleading in this case because it includes problems that could not be solved within the time bound by NONLIN and SNLP and so it underestimates their search depth.

The overall search depth is composed of a number of factors described in the analysis, which includes the fraction of the preconditions considered, the average number of times each precondition is visited, and the average
number of threats detected by each algorithm. Figure 5 shows the fraction of preconditions considered. This number should be one for both SNLP and NONLIN but again the graphs are distorted by the fact that these two systems did not complete all of the problems within the time limit. In that case, there are no number of preconditions of operators that had yet been considered. Note that for most of the problems, TWEAK only expanded roughly 60-80% of the preconditions and as the problems had fewer positive interactions, it was forced to expand more and more of the preconditions.

Figure 6 shows the average of number of times each precondition is visited. As predicted, SNLP and NONLIN visit every precondition exactly once, while TWEAK visits some preconditions more than once. As the number of negative interactions increase, the value for TWEAK increases because it does not protect the conditions that have already been achieved.

Figure 7 shows the average number of threats detected by each of the systems. The fact that SNLP detects a much larger number of threats than both NONLIN and TWEAK comes as no surprise. However, the fact that the number of threats detected by NONLIN is less than the number detected by TWEAK when \( R \ll 1 \) was not predicted by the analysis. This appears to be due to the fact that the negative threats that NONLIN protects against impose additional ordering constraints on the plan and a more linearly ordered plan has fewer potential threats.

4.2.3 Average CPU Time

The average CPU time for solving problems in the artificial domain is shown in Figure 8. The result fits exactly with our predictions. One thing to note is that no system performs absolutely the best throughout the entire spectrum defined by \( R \). Another is that although NONLIN did well as compared to SNLP when \( R \) is small, it is never significantly better than SNLP. In the case where it does outperform SNLP it is dramatically worse than TWEAK. This effect should lend credibility to the protection against positive threats as used in SNLP. Although protection of positive threats seemed clumsy when \( R \) is small, when the number of negative threats is relatively large the protection method used by SNLP imposes more constraints on a plan. The resulting plans in SNLP's search space are more linear due to the additional constraints. The computational advantage of dealing with a more linear plan compensates for the loss of efficiency due to the protection of positive threats.
5 Related Work and Conclusions

As we stated in the introduction, little work has been done on comparing different partial order planners. An exception is the work by Kambhampati (Kambhampati, 1993; Kambhampati, 1992], who (concurrently with our work) carried out a set of experiments to test the merits of different partial-order planners. In that work, a pair of partial-order planners MP and MP-I are proposed that build upon SNLP and NONLIN by making use of multiple contributors to achieve a precondition. Experiments in a set of closely related domains were conducted, and the resulting comparison of SNLP, NONLIN, TWEAK, MP, and MP-I show that MP-I outperforms all of the rest, and that NONLIN in one test performed much better than both SNLP and TWEAK (Figure 8, [Kambhampati, 1993]).

Contrasting Kambhampati's results to ours, we note that the former is based on a fixed domain. Our results clearly demonstrate that varying the ratio \( R \) of positive to negative threats experienced by a planner, almost any comparison result can be obtained; when \( R \ll 1 \) the comparison results should be dramatically different from that when \( R \gg 1 \). Thus, it is not surprising that one can find a domain, with a specific \( R \) value, where SNLP and/or TWEAK perform worse than NONLIN. From this perspective, the work by Kambhampati can be seen as orthogonal to ours; while we search for domain features by which to determine the relative performance of each system, Kambhampati looks for the best planner on a single point in the spectrum of features.

In summary, we have presented a detailed comparison of the goal protection strategies used in the SNLP, NONLIN, and TWEAK planning algorithms. The analysis provides a foundation for predicting the conditions under which different planning algorithms will perform well. As the results show, SNLP and NONLIN perform better than TWEAK when the ratio of negative threats to positive threats is large, and TWEAK performs significantly better than SNLP and NONLIN in the opposite case. The implications of these results for someone building a practical planning system is that the most appropriate goal protection strategy depends on the characteristics of the problem being solved. This paper provides an important step in building useful planners by identifying a feature of planning domains that has a major impact on the performance of different planning algorithms.

References


