Design Tradeoffs in Partial Order (Plan Space) Planning

Subbarao Kambhampati*
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406

Abstract
Despite the long history of classical planning, there has been very little comparative analysis of the performance tradeoffs offered by the multitude of existing planning algorithms. This is partly due to the many different vocabularies within which planning algorithms are usually expressed. In this paper I provide a generalized algorithm for refinement planning, and show that planners that search in the space of (partial) plans are specific instantiations of this algorithm. The different design choices in partial order planning correspond to the different ways of instantiating the generalized algorithm. I will analyze how these choices affect the search-space size and refinement cost of the resultant planner. Finally, I will concentrate on two specific design choices, viz., protection strategies and tractability refinements, and develop some hypotheses regarding the effect of these choices on the performance on practical problems. I will support these hypotheses with a focused empirical study.

1 Introduction
The idea of generating plans by searching in the space of (partially ordered or totally ordered) plans has been around for almost twenty years, and has received a lot of formalization in the past few years. Much of this formalization has however been limited to providing semantics for plans and actions, and proving soundness and completeness of planning algorithms. There has been very little effort directed towards comparative analysis of the performance tradeoffs offered by the multitude of plan-space planning algorithms. An important reason for this state of affairs is the seemingly different vocabularies and/or frameworks within which many of the algorithms are usually expressed.

The primary purpose of this paper is to explicate the design choices in plan-space planners and understand the tradeoffs offered by them. To this end, I provide a generalized refinement planning algorithm, Generalized Plan, that describes the various ways in which its individual steps can be instantiated; and explain how the various instantiations occur the existing plan-space planners. I will then develop models for analyzing the effect of different design choices, i.e., the different ways of instantiating Generalized Plan, on the search space size and refinement cost. Finally, I will demonstrate the practical utility of this analysis by using it to predict the relative performance differentials caused by the various design choices. I will evaluate these predictions with the help of empirical comparisons between five normalized instantiations of Generalized Plan.

The paper is organized as follows: Section 2 discusses a unified representation and semantics for partial plans. Section 3 presents the generalized refinement planning algorithm and describes the different ways of instantiating its components. Section 4 describes the effect of different instantiation choices on the search space size and refinement cost of the resulting planning algorithm. Section 5 considers the effect of various design choices on practical performance. Section 6 presents the conclusions.

2 Preliminaries
A ground operator sequence $S: o_1, o_2, \ldots, o_n$ is said to be a solution to a planning problem $(I, G)$, where $I$ is the initial state of the world, and $G$ is the goal expression, if $S$ can be executed starting from $I$, and the resulting state will satisfy the goal expression. A solution $S$ is said to be minimal if no operator sequence obtained by removing some operators from $S$ can also be a solution. A planning strategy is said to be complete if it can find all the minimal solutions for any given problem. We will be assuming that the domain operators are described in the ADL representation with Precondition and Effect formulas. The precondition and effect formulas are function-free first order predicate logic sentences involving conjunction, negation and quantification. The subset of this representation where both formulas can be represented as conjunctions of function-free first order literals, and all the variables have infinite domains, is called the TWEAK representation (c.f. [2, 11]). We shall restrict our discussion to TWEAK representation wherever possible to keep the exposition simple.

Plan space planners search in the space of partial plans. In [8, 6], I argue that in refinement planning, partial plans are best seen as constraint sets that implicitly represent all the ground operator sequences (called candidates) that are consistent with those constraints. The goal of refinement planning is then to split these candidate sets until a solution candidate can be picked up in bounded time. The following 5-tuple provides a uniform representation for partial plans that is applicable across all refinement planners: $\langle T, O, B, ST, \alpha \rangle$ where $T$ is the set of actions (step-names) in the plan; $T$ contains two distinguished step names $s_0$ and $s_\infty$. $ST$ is a symbol table, which maps step names to domain operators. The special step $s_0$ is always mapped to the dummy operator start, and similarly $s_\infty$ is always mapped to finish. The effects of start and the preconditions of finish correspond, respectively, to the initial state and the desired goals (of attainment) of the planning problem. $O$ is a partial ordering relation over $T$. $B$ is a set of codesignation (binding) and non-codesignation (prohibited bindings) constraints on the variables appearing in the preconditions and post-conditions of the operators. $C$ is a set of auxiliary constraints (see below).

A ground operator sequence $S$ is consistent with the step, ordering and binding constraints of the plan $P$ as long as it contains all the steps of the partial plan, in an order and instantiation consistent with $O$ and $B$. For $S$ to be a candidate of $P$, it should also satisfy the set of auxiliary constraints specified by $C$. An important type of auxiliary
constraint is an interval preservation constraint (IPC), which is specified as a 3-tuple: \((e, p, s')\). A ground operator sequence is said to satisfy an IPC \((e, p, s')\) of a plan \(P\) if there exists a mapping \(M\) between the steps of \(P\) and the elements of \(S\), such that \(M\) is consistent with \(ST\) and every operator of \(S\) that comes between \(M(s)\) and \(M(s')\) preserves \(e\).

To illustrate the above definitions, consider the partial plan \(P_M\) given by the constraint set:

\[
P_M : \{\{t_0, t_1, t_2, t_{\text{end}}\}, \{t_0 < t_1, t_1 < t_2, t_2 < t_{\text{end}}\}, \emptyset, (t_1 \rightarrow t_2, t_2 \rightarrow t_{\text{end}}), (t_0 \rightarrow \text{start}, t_{\text{end}} \rightarrow \text{fin})\}
\]

Let \(S\) be the ground operator sequence \(o_1o_2\), \(S\) is a candidate of \(P_M\) because it contains all the steps of \(P_M\) (under the mapping \(ST\)) in the order consistent with the ordering constraints of \(P_M\). Further, since there are no steps between \(o_1\) and \(o_2\), or \(o_2\) and the end of \(S\), the two interval preservation constraints are also satisfied. By similar arguments, the ground operator sequence \(o_1o_2o_3o_4\) is also a candidate as long as \(o_3\) preserves (does not delete) \(P\) and \(o_4\) preserves \(q\).

A ground linearization (aka completion) of a partial plan \(P : (T, O, B, ST, C)\) is a fully instantiated total ordering of the steps of \(P\) that is consistent with \(O\) (i.e., a topological sort) and \(B\). A ground linearization of a plan is said to be a safe ground linearization if and only if the ground operator sequence corresponding to it (modulo mapping \(ST\)) satisfies all the auxiliary constraints of the plan. For the example plan \(P_M\) discussed above, \(o_1o_2o_3o_4\) is the only ground linearization, and it is also a safe ground linearization (since \(o_1o_2\) satisfies the auxiliary constraints of \(P_M\)).

A partial plan is said to be inconsistent if its candidate set is empty (i.e., there exists no ground operator sequence that is consistent with the constraints of the plan). It can be shown that a partial plan is consistent if it has at least one safe ground linearization, and inconsistent otherwise.

3. A generalized algorithm for Partial-order Planning

The procedure \texttt{Refine-Plan} in Figure 1 describes a generalized refinement-planning algorithm, the specific instantiations of which cover most of the existing partial-order (plan-space) planners.

Given a planning problem \((I, G)\), where \(I\) is the initial state specification and \(G\) is a set of goals (of attachment), the planning process is initiated by invoking \texttt{Refine-Plan} with the null partial plan \(P_0\) where

\[
P_0 : \{(t_0, t_{\text{end}}), \{t_0 \rightarrow t_{\text{start}}, t_{\text{end}} \rightarrow \text{fin}\}, \emptyset\}
\]

Table 1 characterizes many well-known planning algorithms as instantiations of \texttt{Refine-Plan}. In the following, I will briefly discuss the individual steps \texttt{Refine-Plan}.

Goal Selection and Establishment: The primary refinement operation in planning is the so-called establishment operation. It selects a precondition \((C, a)\) of the plan (where \(C\) is a precondition of a step \(a\)), and refines (i.e., adds constraints to) the partial plan such that different steps act as contributors of \(C\) to in different refinements. Pfenzeu [17] provides a general theory of establishment refinement for plans containing actions with conditional and quantified effects. Symmetrically, each refinement corresponds to adding different sets of new step, ordering and binding constraints (as well as additional second order preconditions, in the case of ADL actions [17]) to the parent plan.

An important exception is the hierarchical task reduction planners, such as SIPE [22], O-Plan [3]. However, see [17] for a discussion of how \texttt{Refine-Plan} can be extended to cover these planners.

Algorithm \texttt{Refine-Plan}(\(P\))\textsuperscript{1} Returns refinements of \(P\) \textsuperscript{1}

Parameters:
(i) \texttt{sol}: Solution constructor function. (ii) \texttt{pick-prec}: the routine for picking the goal to be established. (iii) \texttt{interacts?}: the routine used by pre-ordering to check if a pair of steps interact. (iv) \texttt{conflict-resolve}: the routine which resolves conflicts with auxiliary candidate constraints.

0. Termination Check: If \texttt{sol}(\(P, G\)) returns a solution candidate, return it, and terminate. If it returns \texttt{fail}, fail. Otherwise, continue.

1. Goal Selection: Using the \texttt{pick-prec} function, pick a goal \((C, a)\) (where \(C\) is a precondition of step \(a\)) from \(P\) to establish. Not a backtracking point.

2.1. Goal Establishment: Non-deterministically select a new or existing establishment step \(s'\) for \((C, a)\). Introduce enough ordering and binding constraints, and secondary preconditions to the plan such that (i) \(s'\) precedes \(s\), (ii) \(s'\) will have an effect \(C\), and (iii) \(C\) will persist until \(a\) (i.e., \(C\) is preserved by all the steps intervening between \(s'\) and \(a\)). Backtrack point; all establishment possibilities need to be considered.

2.2. Book Keeping: (Optional) Add auxiliary constraints noting the establishment decisions, to ensure that these decisions are protected by any later refinements. This in turn reduces the redundancy in the search space. The protection strategies may be one of goal protection, interval protection and contributor protection (see text). The auxiliary constraints may be one of point truth constraints or interval preservation constraints.

3. Tractability Refinements: (Optional) These refinements help in making the planning handling and consistency check tractable. Use either one or both:

3.a. Pre-Ordering: Impose additional orderings between every pair of steps of the partial plan that possibly interact according to the static interaction matrix \texttt{interacts?}. Backtrack point; all interaction orderings need to be considered.

3.b. Conflict Resolution: Add orderings, bindings and/or secondary (preservation) preconditions to resolve conflicts between the steps of the plan, and the plan’s auxiliary candidate constraints. Backtrack point; all possible conflict resolution constraints need to be considered.

4. Consistency Check: (Optional) If the partial plan is inconsistent (i.e., has no safe ground linearizations), fail. Else, continue.

5. Recursive Invocation: Recursively invoke \texttt{Refine-Plan} on the refined plan.

Figure 1: A generalized refinement algorithm for plan-space planning

The strategy used to select the particular precondition \((C, a)\) to be established (called goal selection strategy) can be arbitrary, demand driven (e.g., select a goal only when it is not necessarily true in all ground linearizations of the current partial plan), or can depend on some ranking based on precondition abstraction [19]. The cost of using each type of goal selection strategy depends on the type of partial plans maintained by the planner (see Table 1). Protecting establishments through book keeping is possible to limit \texttt{Refine-Plan} to establishment refinements alone and still get a sound and complete planner. Chapman’s Tweak [2] is such a planner. However, such a planner is not guaranteed to respect its previous establishment decisions while making new ones, and thus may do a lot of redundant work in the worst case. The book-keeping step attempts to reduce this redundancy by posting auxiliary constraints on the partial plan to protect the establishments.

The protection strategies used by classical partial order planners
From: AIPS 1994 Proceedings. Copyright © 1994, AAAI (www.aaai.org). All rights reserved.

Table 1: Characterization of existing planners as instantiations of Refine-Plan. The last three planners have not been described in the literature previously. They are used in Section 5 to facilitate normalized comparisons.

<table>
<thead>
<tr>
<th>Planner</th>
<th>Solo Constructor</th>
<th>Goal Selection</th>
<th>Book-keeping</th>
<th>Tractability Refinements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tweak [2]</td>
<td>MTC-based O(n^2)</td>
<td>MTC-based O(n^2)</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>UA [15]</td>
<td>MTC-based O(n^2)</td>
<td>MTC-based O(n^2)</td>
<td>None</td>
<td>Unambiguous ordering</td>
</tr>
<tr>
<td>Nomin [21]</td>
<td>MTC (Q&amp;A) based</td>
<td>Arbitrary O(1)</td>
<td>Interval &amp; Goal Protection via Q&amp;A</td>
<td>Conflict Resolution</td>
</tr>
<tr>
<td>TOCL [1]</td>
<td>Protection based O(1)</td>
<td>Arbitrary O(1)</td>
<td>Contributor protection</td>
<td>Total ordering</td>
</tr>
<tr>
<td>Pedestal [13]</td>
<td>Protection based O(1)</td>
<td>Arbitrary O(1)</td>
<td>Interval Protection</td>
<td>Total ordering</td>
</tr>
<tr>
<td>SNLP [14]</td>
<td>Protection based</td>
<td>Arbitrary</td>
<td>Contributor protection</td>
<td>Conflict resolution</td>
</tr>
<tr>
<td>UCOP [18]</td>
<td>O(1)</td>
<td>O(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP, MP-I [9]</td>
<td>Protection based</td>
<td>Arbitrary</td>
<td>(Multi) contributor protection</td>
<td>Conflict resolution</td>
</tr>
<tr>
<td>SNLP-UA (Sec. 5)</td>
<td>MTC based O(n^2)</td>
<td>MTC based O(n^2)</td>
<td>Contributor protection</td>
<td>Unambiguous Ordering</td>
</tr>
<tr>
<td>SNLP-MTC</td>
<td>MTC based O(n^2)</td>
<td>MTC based O(n^2)</td>
<td>Contributor protection</td>
<td>Conflict resolution</td>
</tr>
<tr>
<td>McNONLIN-MTC</td>
<td>MTC based O(n^2)</td>
<td>MTC based O(n^2)</td>
<td>Interval protection</td>
<td>Conflict resolution</td>
</tr>
</tbody>
</table>

4 Modeling and Analysis of Design Tradeoffs

Table 1 characterizes many of the well known plan-space planning algorithms as instantiations of the Refine-Plan algorithm. Understanding the performance tradeoffs offered by the various planners is thus reduced to understanding the ramifications of instantiating the Refine-Plan algorithm in different ways. In this Section, I discuss how these instantiation choices affect the search space size and the refinement cost (the per-invocation cost of Refine-Plan) of the resulting planners.

I will start by developing two complementary models for the size of the search space explored by Refine-Plan in a breadth-first search regime. Suppose $F_a$ is the $d^{th}$ level fringe of the search tree explored by Refine-Plan. Let $\mu_a$ be the average size of the candidate sets of the partial plans in the $d^{th}$ level fringe, and $\mu(2)$ be the redundancy factor, i.e., the average number of partial plans on the fringe whose candidate sets contain a given candidate in $\mathbb{K}$ (see Section 2). It is easy to see that $|F_a| \times \mu_a = |\mathbb{K}| \times \mu(2)$ (where $|.|$ is used to denote the cardinality of a set). If $b$ is the average branching factor of the search, then the size of $d^{th}$ level fringe is also given by

\[ |F_a| \times \mu_a = |\mathbb{K}| \times \mu(2) \]

Note that solution constructor function may also return a "fail" on a given partial plan. The difference between this and the consistency check is that the latter fails only when the partial plan has an empty candidate set, while the solution constructor can fail as long as the candidate set of the partial plan does not contain any solutions to the given problem.

The solution constructors discussed above are all conservative in that they require that all safe ground linearizations of a partial plan be solutions. In [8], we show that it is possible to provide polynomial $k$-eager solution constructors, which randomly check if safe ground linearizations of the plan to see if any of them are solutions. These constructors are sound and complete and are guaranteed to terminate the search before the conservative constructor functions.

Although both $|\mathbb{K}|$ and $\mu(2)$ can be infinite, by restricting our attention to maximal solutions, it is possible to construct finite versions of both. Given a planning problem instance $P$, let $l_{max}$ be the length of the longest ground operator sequence that is a minimal solution of $P$. We can now define $\mathbb{K}$ to be the set of all ground operator sequences of up to length $l_{max}$. Similarly, we can redefine the candidate set of a partial plan to consist only of the subset of its candidates that are not longer than $l_{max}$.
the candidate sets, driving down useless activity and populates the search fringe with plans with empty \( P \). Consistency Check: As mentioned (of.

Consistency is NP-hard even for ground plans in TWEAK representation. Thus, the size of the fringe of the search will never be greater than the size of the candidate space \( |K| \). Such a search is called a strong systematic search. In terms of the refinement cost,

\[
|F_d| \approx O(b^4) \approx \frac{|K| \times \rho_d}{\kappa_d}.
\]

The average branching factor, \( b \), can be split into two components, \( b_1 \), the establishment branching factor, and \( b_2 \), the tractability refinement branching factor, such that \( b = b_1 \times b_2 \). \( b_1 \) and \( b_2 \) correspond, respectively, to the branching made in steps 2 and 3 of the Refine-Plan algorithm.

If \( C \) is the average cost per invocation of the Refine-Plan algorithm, and \( d_n \) is the effective depth of the search, then the cost of the plan is \( C \times |F_d| \). \( C \) itself can be decomposed into three main components: \( C = C_s + C_2 + C_s \), where \( C_s \) is the establishment cost (including the cost of selecting the goal to work on), \( C_2 \) is the cost of solution constructor, and \( C_s \) is the cost of consistency check.

Armed with this model of the search space size and refinement cost, we shall now look at the effect of the various ways of instantiating each step of the Refine-Plan algorithm on the search space size and the cost of refinement.

Solution Constructor: Stronger solution constructors allow the search to end earlier, reducing the effective depth of the search, and thereby the size of the explored search space. In terms of candidate space view, stronger solution constructors lead to larger \( \kappa_d \) at the termination fringe. However, at the same time they increase the cost of refinement \( C \) (specifically the \( C_s \) factor).

Book Keeping: Addition of book-keeping techniques tend to reduce the redundancy factor \( \rho_d \). In particular, use of contributor protection makes the search systematic, eliminating all the redundancy in the search space and making \( \rho_d \) equal to 1 \([14, 16]\). This tends to reduce the fringe size, \( |F_d| \). Book keeping constraints do however tend to increase the cost of consistency check. In particular, checking the consistency of a partial plan containing interval preservation constraints is NP-hard even for ground plans in TWEAK representation \((\text{c.f.} [20])\).

Consistency Check: As mentioned earlier, the motivation behind consistency check is to avoid refining inconsistent plans (or the plans with empty candidate sets). Refining inconsistent plans is a useless activity and populates the search fringe with plans with empty candidate sets, driving down \( \kappa_d \). The stronger the consistency check, the smaller this reduction. In particular, if the planner uses a sound and complete consistency check that is capable of identifying every inconsistent plan, then the average candidate set \( \kappa_d \) is guaranteed to be greater than or equal to 1. Combined with a systematic search, this will guarantee that the fringe size of the search will never be greater than the size of the candidate space \( |K| \).

5 Empirical Analysis of Performance Tradeoffs

In this section, I will evaluate the utility of the tradeoffs model developed in the previous section in predicting and explaining empirical performance. From Table 1, we note that two prominent dimensions of variation among existing planners are the bookkeeping strategies and tractability refinements they use. In the interests of space, we shall restrict our attention to the performance tradeoffs offered by these dimensions of variation.

An important prerequisite for any such comparative analysis is to normalize the effects of other dimensions of variation. Our generalized algorithm provides a systematic basis for doing such a normalization. In particular, I will consider five instances of Refine-Plan, called SNLP-MTC, UA, SNLP-UA, TWEAK and McNONLIN-MTC, which differ only in the book-keeping strategies and the tractability refinements they use. All these planners use the same MTC-based goal selection strategy, and MTC-based solution constructor function (see Section 3). SNLP-MTC and SNLP-UA use contributor protection, while McNonlin-MTC uses interval protection \([6, 9]\). UA and SNLP-UA use unambiguous preordering strategies (SNLP-UA uses a stronger notion of interaction which makes two steps interact even if they share add list literals \([6]\)). SNLP-MTC and McNonLIN-MTC use conflict resolution strategies as their tractability refinements.

5.1 Tractability Refinements

From the discussion in Section 4, we note that presence of tractability refinements increases \( b_2 \). Specifically, we can see that for the

Kambhampati 95
The first two plots compare average branching factors in ART-MD-RD experiments. The third plot shows the effect of misdirection on protection strategies.

Figure 3 shows the performance of the five planners in ART-MD-RD experiments. The plots compare average branching factors for two different goal orderings: MD-RD and FIFO. The plots illustrate the impact of protection strategies on performance.

To evaluate this hypothesis, I compared the performance of the five planners on problems from an artificial domain called ART-MD-RD [10], which is specified as:

\[ A_{i \text{ prec}} : I_i, \text{he add} : G_i, \text{he del} : \{j \mid j < i \} \cup \{\text{he}\} \text{ for even } i \]

\[ A_{i \text{ prec}} : I_i, \text{he add} : G_i, \text{he del} : \{j \mid j < i \} \cup \{\text{he}\} \text{ for odd } i \]

The plots in Figure 2 show the performance of the five planners on problems from this domain. Each point in the plot corresponds to an average over 30 random problems with the given number of goals (drawn from \( \{G_1, \ldots, G_6\} \)). The initial state is the same for all the problems, and contains \( \{I_1, \ldots, I_6\} \cup \{\text{he}\} \). The plots show the performances for two different goal orderings (over and above the MTC-based goal selection strategy). In LIFO ordering, a goal and all its subgoals (which are not necessarily true according to MTC) are established before the next higher level goal is addressed. In the FIFO ordering, all the top level goals are established before their subgoals are addressed. All the planners use number of steps as the heuristic.

If the performance depended only on the \( b_h \) factor, we would expect to see planners with less eager tractability refinements perform better. We note that although our hypothesis ignores possible secondary interactions between tractability refinements and other parts of the ReFine-Plan algorithm, one such interaction is between tractability refinement and establishment refinement. Specifically, the additional linearization of the plan done at the tractability refinement stage may sometimes wind up reducing the number of possible establishment refinements for the goals considered in later stages. This could, in turn reduce \( b_h \) and sometimes even \( b_y \). Thus, the overall effect of tractability refinements on the search space thus depends on whether or not the increase in

Table 2: Estimates of average redundancy factor and average candidate candidate set size at the termination fringe for 30 random 5-goal problems in ART-MD-RD domain

<table>
<thead>
<tr>
<th>Planner</th>
<th>LIFO</th>
<th>FIFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tweak</td>
<td>2.93</td>
<td>3.0</td>
</tr>
<tr>
<td>UA</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>McNonlin-MTC</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>SNLP-MTC</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>SNLP-UA</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
The lack of direct correlation between protection strategies and performance should not be surprising. As shown in [10], the redundancy in the search space is expected to affect performance only when the planner is forced to examine a significant part of its search space before finding the solution. Thus, we expect that the differences in protection strategies alone will affect the performance only when the apparent solution density is low enough to force the planner to search a substantial part of its search space.

To evaluate this hypothesis further, I compared TWEAK and SNLP-MTC in a variant of ART-MD-RD domain without the $k_f/k_s$ conditions (similar to $D^S$) domain described in [12], where their average performances are similar. Both planners were started off with a mingoal heuristic, which ranks a partial plan by the number of outstanding open-constraints (i.e., preconditions that are not necessarily true according to MTC). I then systematically varied the probability $p$ (called the misdirection parameter) with which both planners will reject the direction recommended by the heuristic and will select the worst ranked branch instead. Assuming that the initial heuristic was a good heuristic for the problem, to a first order of approximation, we would expect that increasing the misdirection parameter degrades the planner's ability to zero-in on the solutions, forcing it to consider larger and larger parts of its search space. By our hypothesis, strong protection strategies should help in such situations.

The third plot in Figure 3 shows the performance of the planners (measured in terms of average cpu time taken for solving a set of 20 random problems), as a function of misdirection parameter. It shows that as the misdirection parameter increases, the performance of TWEAK, which employs no protections, degrades much more drastically than that of SNLP-MTC, which employs contributor protection. These results thus support our hypothesis.

5.3 Experimental Conclusions

While our experiments involved only two artificial domains, they do bring out some general patterns regarding the effect of tractability refinements and protection strategies on performance. They show that the empirical performance differentials between the five planners are determined to a large extent by the differences in the tractability refinements than by the differences in protection strategies. The protection strategies themselves only act as an insurance policy that pays off in the worst-case scenario when the planner is forced to look at a substantial part of its search space.

Further, although eager tractability refinements will degrade the performance in general, they may sometimes improve the performance because of the indirect interaction between $h_t$ and $h_a$. Two factors that are predictive of the $h_t \rightarrow h_a$ interaction are (i) the presence of high-frequency conditions (i.e., conditions which are established and deleted by many actions in the domain) and (ii) whether the high-frequency conditions are selected for establishment towards the beginning or the end of the planning process. The experiments also show that the performance of planners using conflict resolution strategies is more stable with respect to the $h_t$ and $h_a$ interaction than that of planners using pre-ordering strategies. This can be explained by the fact that the former are more sensitive to the role played by the steps in the current partial plan than the latter.

The primary contribution of this paper is a unified framework for understanding and analyzing the design tradeoffs in partial-order plan-space planning. I started by developing a generic refinement planning algorithm in Section 3, and showed that most existing planner instantiations of this algorithm. I then developed a model for estimating the refinement cost and search space size of a planner and discussed how they are affected by the different design choices. Finally, I have described some preliminary empirical studies aimed at understanding the effect of two of these choices—tractability refinements and protection strategies—on the relative performance of different planners. These studies show that the performance is affected more by the differences in tractability refinements than by the differences in protection strategies. While this paper makes an important step towards understanding of the comparative performance of partial-order planners, further work is still needed to develop a predictive understanding of which instantiations of a planner will perform best in which types of domains. Finally, although I concentrated on partial order planners, in [7] I show that a planner can also be extended to cover task reduction planners.

6 Conclusion

The primary contribution of this paper is a unified framework for understanding and analyzing the design tradeoffs in partial-order plan-space planning. I started by developing a generic refinement planning algorithm in Section 3, and showed that most existing planner instantiations of this algorithm. I then developed a model for estimating the refinement cost and search space size of a planner and discussed how they are affected by the different design choices. Finally, I have described some preliminary empirical studies aimed at understanding the effect of two of these choices—tractability refinements and protection strategies—on the relative performance of different planners. These studies show that the performance is affected more by the differences in tractability refinements than by the differences in protection strategies. While this paper makes an important step towards understanding of the comparative performance of partial-order planners, further work is still needed to develop a predictive understanding of which instantiations of a planner will perform best in which types of domains. Finally, although I concentrated on partial order planners, in [7] I show that a planner can also be extended to cover task reduction planners.

References