To appear in Proc. 2nd Intl. Conf. on AI Planning Systems (AIPS-94) May, 1994

Design Tradeoffs in Partial Order (Plan Space) Planning

Subbarao Kambhampati*
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406
Email: rao@asuvax.asu.edu

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 focussed 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.[1] 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, `Refine-Plan`; describe the various ways in which its individual steps can be instantiated; and explain how the various instantiations cover 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, `Refine-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 `Refine-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.

*This research is supported in part by National Science Foundation under grant IRI-9210997, and ARPA/Rome Laboratory planning initiative under grant F30602-93-C-0039. Special thanks to David McAllester and Bulusu Gopi Kumar for constructive suggestions in the initial stages of this work, and Dan Weld, Craig Knoblock, Qiang Yang and Eric Jacopin for helpful discussions.

[1] The work of Barrett and Weld [1] as well as Minton et. al. [15] are certainly steps in the right direction. However, they do not tell the full story since the comparison there was between a specific partial order and total order planner. The comparison between different partial order planners itself is still largely unexplored.

2 Preliminaries

A ground operator sequence $S : o_1, o_2, \cdots o_n$ is said to be a **solution** to a planning problem $[\mathcal{I}, \mathcal{G}]$, where $\mathcal{I}$ is the initial state of the world, and $\mathcal{G}$ is the goal expression, if S can be executed starting from $\mathcal{I}$, 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 [17] 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: $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ where: T is the set of actions (step-names) in the plan; T contains two distinguished step names t_0 and t_∞. $\mathcal{ST}$ is a symbol table, which maps step names to domain operators. The special step t_0 is always mapped to the dummy operator `start`, and similarly t_∞ 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. $\mathcal{B}$ 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. $\mathcal{L}$ 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 $\mathcal{P}$ as long as it contains all the steps of the partial plan, in an order and instantiation consistent with O and $\mathcal{B}$. For S to be a candidate of $\mathcal{P}$, it should also satisfy the set of auxiliary constraints specified by $\mathcal{L}$.[2] An important type of auxiliary

[2] For a more formal treatment of the semantics of candidate sets and the auxiliary constraints, see [8]. There, I differentiate between two types of auxiliary constraints-- auxiliary candidate constraints, which need to be

constraint is an **interval preservation constraint** (IPC), which is specified as a 3-tuple: $\langle s, p, s' \rangle$. A ground operator sequence is said to satisfy an IPC $\langle s, p, s' \rangle$ of a plan $\mathcal{P}$,if there exists a mapping $\mathcal{M}$ between the steps of $\mathcal{P}$ and the elements of S, such that $\mathcal{M}$ is consistent with $\mathcal{ST}$ and every operator of S that comes between $\mathcal{M}[s]$ and $\mathcal{M}[s']$ preserves p.

To illustrate the above definitions, consider the partial plan $\mathcal{P}_E$ given by the constraint set:

$$\mathcal{P}_E : \left\langle \begin{array}{l} \{t_0, t_1, t_2, t_\infty\}, \{t_0 \prec t_1, t_1 \prec t_2, t_2 \prec t_\infty\}, \emptyset, \\ \{t_1 \rightarrow o_1, t_2 \rightarrow o_2, t_0 \rightarrow \texttt{start}, t_\infty \rightarrow \texttt{fin}\}, \\ \{\langle t_1, p, t_2 \rangle, \langle t_2, q, t_\infty \rangle\} \end{array} \right\rangle$$

Let S be the ground operator sequence $o_1 o_2$. S is a candidate of $\mathcal{P}_E$ because it contains all the steps of $\mathcal{P}_E$ (under the mapping $\mathcal{ST}$) in the order consistent with the ordering constraints of $\mathcal{P}_E$. 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 $o_1 o_3 o_2 o_4$ 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 $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ is a fully instantiated total ordering of the steps of $\mathcal{P}$ that is consistent with O (i.e., a topological sort) and $\mathcal{B}$. 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 $\mathcal{P}_E$ discussed above, $t_0 t_1 t_2 t_\infty$ is the only ground linearization, and it is also a safe ground linearization (since $o_1 o_2$ satisfies the auxiliary constraints of $\mathcal{P}_E$).

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 [8] 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 `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.[3]

Given a planning problem [I,G], where I is the initial state specification and G is a set of goals (of attainment), the planning process is initiated by invoking `Refine-Plan` with the null partial plan $\mathcal{P}_\emptyset$ where

$$\mathcal{P}_\emptyset : \langle \{t_0, t_\infty\}, \{t_0 \prec t_\infty\}, \emptyset, \{t_0 \rightarrow \texttt{start}, t_\infty \rightarrow \texttt{fin}\}, \emptyset \rangle$$

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

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

satisfied by every candidate of the plan, and auxiliary solution constraints which need to be satisfied only by those candidates of the plan that are solutions to the planning problem. To keep things simple, I am assuming that all the auxiliary constraints are auxiliary candidate constraints.

[3] An important exception is the hierarchical task reduction planners, such as SIPE [22], O-Plan [3]. However, see [7] for a discussion of how `Refine-Plan` can be extended to cover these planners.

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

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

1. Goal Selection: Using the `pick-prec` function, pick a goal $\langle C, s \rangle$ (where C is a precondition of step s) from $\mathcal{P}$ to establish. *Not a backtrack point.*

2.1. Goal Establishment: Non-deterministically select a new or existing establisher step s' for $\langle C, s \rangle$. 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 s (i.e., C is preserved by all the steps intervening between s' and s). *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 plan 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 metric `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 `Refine-plan` on the refined plan.

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

The strategy used to select the particular precondition $\langle C, s \rangle$ 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: It is possible to limit `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

Planner	Soln. Constructor	Goal Selection	Book-keeping	Tractability Refinements
Tweak [2]	MTC-based $O(n^4)$	MTC-based $O(n^4)$	None	None
UA [15]	MTC-based $O(n^4)$	MTC-based $O(n^4)$	None	Unambiguous ordering
Nonlin [21]	MTC (Q&A) based	Arbitrary $O(1)$	Interval & Goal Protection via Q&A	Conflict Resolution
TOCL [1]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Total ordering
Pedestal [13]	Protection based $O(1)$	Arbitrary $O(1)$	Interval Protection	Total ordering
SNLP [14] UCPOP [18]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Conflict resolution
MP, MP-I [9]	Protection based	Arbitrary	(Multi) contributor protection	Conflict resolution
SNLP-UA (Sec. 5)	MTC based $O(n^4)$	MTC based $O(n^4)$	Contributor protection	Unambiguous Ordering
SNLP-MTC	MTC based $O(n^4)$	MTC based $O(n^4)$	Contributor protection	conflict resolution
McNONLIN-MTC	MTC based $O(n^4)$	MTC based $O(n^4)$	Interval protection	conflict resolution

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.

come in two main varieties: interval protection (aka causal link protection, or protection intervals), and contributor protection (aka exhaustive causal link protection [9]). They can both be represented in terms of the interval preservation constraints.

Suppose the planner just established a condition c at step s with the help of the effects of the step s'. For planners using interval protection (e.g., PEDESTAL [13]), the book-keeping constraint requires that no candidate of the partial plan can have p *deleted* between operators corresponding to s' and s. It can thus be modeled in terms of interval preservation constraint $\langle s', p, s \rangle$. Finally, for book keeping based on contributor protection, the auxiliary constraint requires that no candidate of the partial plan can have p either *added* or deleted between operators corresponding to s' and s.[4] This contributor protection can be modeled in terms of the twin interval preservation constraints $\langle s', p, s \rangle$ and $\langle s', \neg p, s \rangle$.[5]

Consistency Check: The aim of the consistency check is to prune inconsistent nodes (i.e., nodes with empty candidate sets) from the search space, thereby improving the performance of the overall refinement search.[6] The consistency check can be done by ensuring that the partial plan has at least one safe ground linearization. This requires checking each ground linearization against all the auxiliary constraints. For general partial orderings with causal-link based auxiliary constraints, the consistency check is NP-hard [20] even for TWEAK action representation.

Pre-ordering and Conflict Resolution to aid tractable refinement: Both preordering and conflict resolution steps aim to make `Refine-Plan` tractable by making the consistency check polynomial. In the case of pre-ordering, this aim is achieved by restricting the type of partial orderings in the plan such that consistency with respect to auxiliary constraints can be checked without explicitly enumerating all the ground linearizations. Two possible pre-ordering techniques are *total ordering* and *unambiguous ordering* [15]. Total ordering orders every pair of steps in the plan, while unambiguous ordering orders a pair of steps only when their preconditions and effects have an overlap (thereby making it possible that the two steps will interact). Both of them guarantee that in the refinements produced by them, either all ground linearizations will be safe or none will be. Thus, consistency can be checked in polynomial time by examining any one ground linearization. In the case of conflict resolution, the partial plan is refined (by adding ordering and binding constraints) until each possible violation of the auxiliary constraint (called conflict) is individually resolved. This ensures that all remaining ground linearizations are safe. Thus, checking the partial plan consistency will amount to checking for the existence of ground linearizations (which can be done by checking ordering and binding consistency).

Solution Constructor function: The job of a solution-constructor function is to look for and return a solution candidate that is consistent with the current constraints of the partial plan.[7] Existing solution constructors fall into two broad categories -- (i) those which use the modal truth criterion (MTC) to check that *all* the safe ground linearizations correspond to solutions and (ii) those that depend upon interval and contributor protection strategies and conflict resolution (see below) to guarantee that all goals are established and that none of the establishments are violated (we call these protection based constructors). [8]

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 $\mathcal{F}_d$ is the d^{th} level fringe of the search tree explored by `Refine-Plan`. Let $\kappa_d \geq 0$ be the average size of the candidate sets of the partial plans in the d^{th} level fringe, and $\rho_d (\geq 1)$ be the redundancy factor, i.e., the average number of partial plans on the fringe whose candidate sets contain a given candidate in $\mathcal{K}$ (see Section 2). It is easy to see that $|\mathcal{F}_d| \times \kappa_d = |\mathcal{K}| \times \rho_d$ (where $|.|$ is used to denote the cardinality of a set).[9] If b is the average branching factor of the search, then the size of d^{th} level fringe is also given by

[4] See [6] for a coherent reconstruction of the ideas underlying goal protection strategies.

[5] Multi-contributor protections, such as those described in [9] can be represented as a disjunction of IPCs.

[6] Thus, from completeness point of view, consistency check is really an optional step.

[7] 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.

[8] 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 k 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.

[9] Although both $|\mathcal{K}|$ and κ_d can be infinite, by restricting our attention to minimal solutions, it is possible to construct finite versions of both. Given a planning problem instance P, let l_m be the length of the longest ground operator sequence that is a minimal solution of P. We can now define $\mathcal{K}$ to be the set of all ground operator sequences of up to length l_m. Similarly, we can redefine the candidate set of a partial plan to consist of only the subset of its candidates that are not longer than l_m.

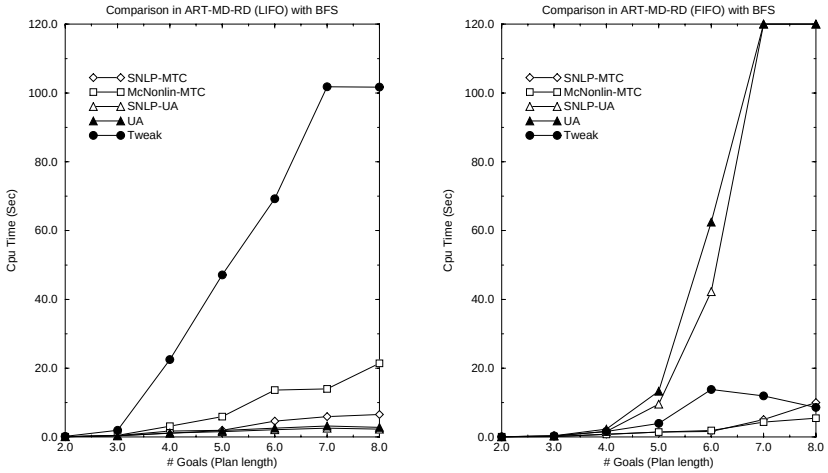

Figure 2: *Comparative performance in ART-MD-RD domain (see text)*

b^d. Thus, we have,

$$|\mathcal{F}_d| \approx O(b^d) \approx \frac{|\mathcal{K}| \times \rho_d}{\kappa_d}.$$

The average branching factor, b can be split into two components, b_e, the establishment branching factor, and b_t the tractability refinement branching factor, such that $b = b_e \times b_t$. b_e and b_t 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_e is the effective depth of the search, then the cost of the planning is $C \times |\mathcal{F}_{d_e}|$. C itself can be decomposed into three main components: $C = C_e + C_c + C_s$, where C_e, is the establishment cost (including the cost of selecting the goal to work on), C_s is the cost of solution constructor, and C_c 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 κ_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 ρ_d. In particular, use of contributor protection makes the search *systematic*, eliminating all the redundancy in the search space and making ρ_d equal to 1 [14, 6]. This tends to reduce the fringe size, $|\mathcal{F}_d|$. 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 (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 κ_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 κ_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 $|\mathcal{K}|$. Such a search is called a *strong systematic search*. In terms of the refinement cost, stronger consistency checks tend to be costlier, thereby driving up the refinement cost (in particular C_s).

Tractability Refinements: The primary motivation for tractability refinements, whether pre-ordering or conflict-resolution, is to make the consistency check tractable. They thus primarily reduce the C_c component of refinement cost. In the case of pre-ordering refinements, they also tend to reduce the cost of goal selection and solution-construction, especially when the latter are based on MTC (thereby reducing the C_s and C_e components) [6]. In terms of search space size, tractability refinements further refine the plans coming out of the establishment stage, increasing the b_t component of the branching factor. While conflict resolution strategies introduce orderings between steps based both on the static description of the steps (such as their effects and preconditions) and the role played by them in the current partial plan, the pre-ordering strategies consider only the static description of the steps. Thus, the b_t increase is typically higher for pre-ordering than for conflict resolution strategies.

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 book-keeping 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. 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_t. Specifically, we can see that for the

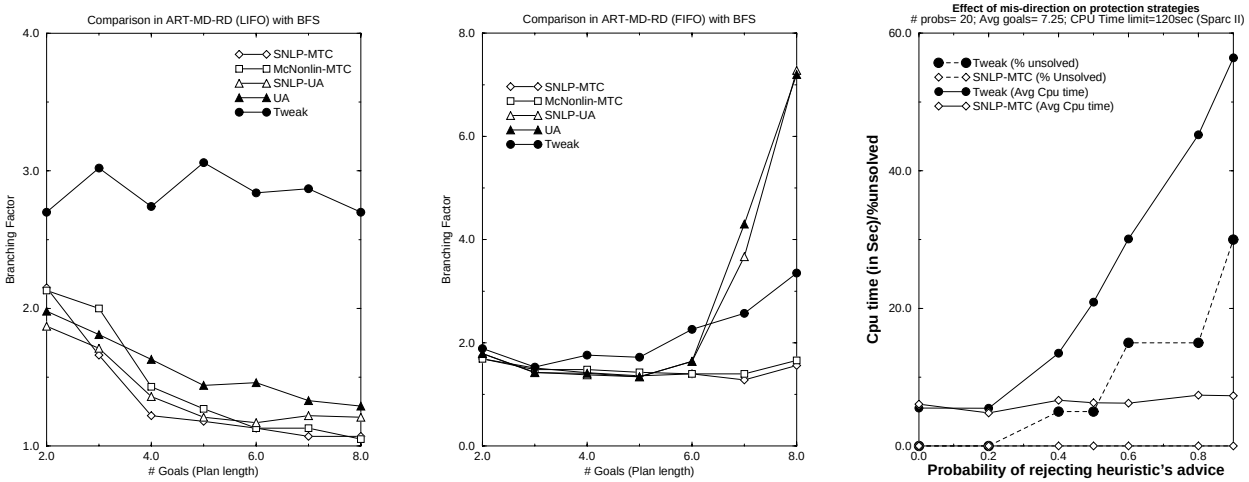

Figure 3: *The first two plots compare average branching factors in ART-MD-RD experiments. The third plot shows the effect of misdirection on protection strategies*

five planners that we are considering, b_t(SNLP-UA) $\geq$ b_t(UA) $\geq$ b_t(SNLP-MTC) $\geq$ b_t(McNONLIN-MTC) $\geq$ b_t(TWEAK). Barring any other interactions, the increase in b_t should increase the average branching factor of the search, and should consequently increase search space size exponentially. Unless the refinement cost is also reduced exponentially (which is not the case for these five planners, and is unlikely in general; Section 4), we would expect planners using more eager tractability refinements (i.e., higher b_t) to perform worse than those using less eager (or no) tractability refinements under breadth-first search regime.

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 $\underline{prec}$: I_i, he $\underline{add}$: G_i, hf $\underline{del}$: $\{I_j | j < i\} \cup \{he\}$ for even i

A_i $\underline{prec}$: I_i, hf $\underline{add}$: G_i, he $\underline{del}$: $\{I_j | j < i\} \cup \{hf\}$ 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 $\{G_1 \cdots G_8\}$). The initial state is the same for all the problems, and contains $\{I_1 \cdots I_8\}$ + $\{he\}$. 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 in the partial plan as the heuristic.

If the performance depended only on the b_t factor, we would expect to see planners with less eager tractability refinements perform better. We note that although our hypothesis holds for FIFO ordering, for LIFO ordering, the observed relative performance is exactly the reverse of what is predicted by this hypothesis. What is more, the plots of average branching factors of the five planners in Figure 3 show that in LIFO ordering, the overall branching factors are opposed to the pattern of b_t.

To understand this apparent discrepancy, we start by noting that 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_e. Since $b = b_e \times b_t$, the overall effect of tractability refinements on the search space thus depends on whether or not the increase in

Planner	LIFO		FIFO	
	ρ_d	κ_d	ρ_d	κ_d
Tweak	1.47	2.93	1.32	30.14
UA	1.76	1.0	1.01	1.0
McNonlin-MTC	1.004	1.007	1.22	34.77
SNLP-MTC	1.0	.77	1.0	13.87
SNLP-UA	1.0	.87	1.0	0.88

Table 2: Estimates of average redundancy factor and average candidate set size at the termination fringe for 30 random 6-goal problems in ART-MD-RD domain

b_t is accompanied by any fortuitous decrease in b_e. (A variant of this type of interaction was first observed by Knoblock and Yang in their experiments [12].)

The behavior of the five planners in LIFO ordering can be explained in terms of the interaction between b_t and b_e. In the case of LIFO ordering, each planner is forced to work on he and hf conditions explicitly. These are *high-frequency* conditions (c.f. [10]) in the sense that there are many actions that can add or delete them. Because of this, the branching factor at the establishment refinement is very high for these conditions, and becomes the dominating factor affecting the performance. Planners using eager tractability refinements have more linearized partial plans and thus offer fewer establishment possibilities than those with less eager tractability refinements. Thus, TWEAK, which does not introduce any tractability orderings, and maintains a more unordered partial plan, incurs the full effect of increased b_e, while the other four planners mitigate it through the linearization introduced through tractability refinement. This explains why the planners with eager tractability refinements perform better.

The situation reverses (and becomes more normal) in the FIFO ordering, where all the top level goals are first addressed before their preconditions. Since G_i, I_i goals are worked on first, and the resulting ordering indirectly establishes hf/he goals, the high-frequency conditions are either not considered for establishment explicitly, or are considered late in the search. Thus, b_e no longer dominates the over all branching factor, and planners with more eager tractability refinement (and higher b_t) perform worse than those with less eager tractability refinement, as expected.

5.2 Book Keeping/Protection Strategies

I will now turn to the effect of protection strategies on performance. From the discussion in Section 4, we note that the primary purpose of the book-keeping strategies is to reduce the redundancy in the search space. Table 2 shows the estimates of the average redundancy factors

at the termination fringe of the five planners for 30 6-goal ART-MD-RD problems.[10] As to be expected, SNLP-MTC and SNLP-UA, which use contributor protections, have no redundancy (ρ_d = 1). However, from the plots in Figure 2, we note that the performance profiles in ART-MD-RD domain are not in correspondence with the redundancy factors. From the same plots, we also note that SNLP-UA, which uses the contributor protection, is closer to UA than SNLP-MTC in performance.

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 hf/he conditions (similar to $D^m S^1$ domain described in [1]), where their average performances are similar. Both planners were started off with a *mingoals* heuristic, which ranks a partial plan by the number of outstanding open-conditions (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 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 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 b_t and b_e. Two factors that are predictive of the $b_t - b_e$ 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 b_t and b_e 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.

[10] The estimates were made by considering the ground operator sequences corresponding to the safe ground linearizations of the plans at the termination fringe.

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 plan-space planners are instantiations of this algorithm. I then developed a model for estimating the refinement cost and search space size of `Refine-Plan` 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 start towards understanding of the comparative performance of partial-order planners, further work is still needed to develop a predictive understanding of which instantiations of `Refine-Plan` will perform best in which types of domains. Finally, although I concentrated on partial order planners, in [7] I show that `Refine-Plan` can also be extended to cover task reduction planners.

References

[1] A. Barrett and D. Weld. Partial Order Planning: Evaluating Possible Efficiency Gains *Artificial Intelligence*, Vol. 67, No. 1, 1994.

[2] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333--377, 1987.

[3] K. Currie and A. Tate. O-Plan: The Open Planning Architecture. *Artificial Intelligence*, 51(1), 1991.

[4] M.G. Georgeff. Planning. In *Readings in Planning*. Morgan Kaufmann, 1990.

[5] J. Jaffar and J. L. Lassez. Constraint logic programming. In *Proceedings of POPL-87,* pages 111--119, 1987.

[6] S. Kambhampati. Planning as Refinement Search: A unified framework for comparative analysis of Search Space Size and Performance. ASU CSE TR 93-004, June, 1993. Available via anonymous ftp from `enws318.eas.asu.edu:pub/rao`

[7] S. Kambhampati. Comparing partial order planning and task reduction planning: A preliminary report. ASU CSE TR-94-001.

[8] S. Kambhampati. Refinement search as a unifying framework for analyzing planning algorithms. In *Proc. 4th Intl. Conf. on Ppls. of KR & R (KR-94)*, May 1994.

[9] S. Kambhampati. Multi-Contributor Causal Structures for Planning: A Formalization and Evaluation. *Artificial Intelligence*. 1994 (To appear)

[10] S. Kambhampati. On the Utility of Systematicity: Understanding tradeoffs between redundancy and commitment in partial order planning In Proceedings of IJCAI-93, Chambery, France, 1993.

[11] S. Kambhampati and D.S. Nau. On the Nature and Role of Modal Truth Criteria in Planning *Artificial Intelligence*, 1994 (To appear).

[12] C. Knoblock and Q. Yang. A Comparison of the SNLP and TWEAK planning algorithms. In Proc. of AAAI Spring Symp. on Foundations of Automatic Planning. March, 1993.

[13] D. McDermott. Regression Planning. *Intl. Jour. Intelligent Systems*, 6:357-416, 1991.

[14] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proc. 9th AAAI*, 1991.

[15] S. Minton, M. Drummond, J. Bresina and A. Philips. Total Order vs. Partial Order Planning: Factors Influencing Performance In *Proc. KR-92*, 1992.

[16] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, Massachusettes (1984).

[17] E.P.D. Pednault. Synthesizing Plans that contain actions with Context-Dependent Effects *Computational Intelligence*, Vol. 4, 356-372 (1988).

[18] J.S. Penberthy and D. Weld. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proc. KR-92*, November 1992.

[19] E. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5(2), 1975.

[20] D.E. Smith and M.A. Peot. Postponing Simple Conflicts in Nonlinear Planning. In *Proc. Eleventh AAAI*, 1993.

[21] A. Tate. Generating Project Networks. In *Proceedings of IJCAI-77*, pages 888--893, Boston, MA, 1977.

[22] D. Wilkins. *Practical Planning*. Morgan Kaufmann (1988).