Planning as Refinement Search: A Unified Framework for Evaluating Design Tradeoffs in Partial-Order Planning

Subbarao Kambhampati [1]

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

Craig A. Knoblock

Information Sciences Institute and Department of Computer Science, University of Southern California, 4676 Admiralty Way, Marina del Rey, CA 90292

Qiang Yang

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

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 we show that refinement search provides a unifying framework within which various planning algorithms can be cast and compared. Specifically, we will develop refinement search semantics for planning, 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. We will analyze how these choices affect the search-space size and refinement cost of the resultant planner, and show that in most cases they trade one for the other. Finally, we 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. We will support these hypotheses with a series of focused empirical studies.

[1] Corresponding author. Fax: (602) 965-2751, E-mail: rao@asu.edu.

1 Introduction

[...] Search is usually given little attention in this field, relegated to a footnote about how ''Backtracking was used when the heuristics didn't work.''
Drew McDermott, [23, p. 413]

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 results for 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. [2] Indeed, there exists a considerable amount of disagreement and confusion about the role and utility of even such long-standing concepts as ''goal protection'', and ''conflict resolution'' -- not to mention the more recent ideas such as ''systematicity.''

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 lack of a unified framework for viewing planning algorithms has hampered comparative analyses and understanding of design tradeoffs, which in turn has severely inhibited fruitful integration of competing approaches.

The primary purpose of this paper is to provide a unified framework for understanding and analyzing the design tradeoffs in partial-order planning. We make five linked contributions:

(i) We provide a unified representation and semantics for partial order planning in terms of refinement search. [3]
(ii) Using these representations, we present a generalized algorithm for refinement planning and show that most existing partial order planners are instantiations of this algorithm.
(iii) The generalized algorithm facilitates the separation of important ideas underlying individual algorithms from ''brand-names'', and thus provides a rational basis for understanding the tradeoffs offered by various planners. We will

[2] The work of Barrett and Weld [2] as well as Minton et al. [26,27] 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. See Section 9 for a more complete discussion of the related work.

[3] Although it has been noted in the literature that most existing classical planning systems are ''refinement planners,'' in that they operate by adding successively more constraints to the the partial plan, without ever retracting any constraint, no formal semantics have ever been developed for planning in terms of refinement search.

characterize the space of design choices in writing partial order planning algorithms as corresponding to the various ways of instantiating the individual steps of the generalized algorithm.

(iv) We will develop a model for estimating the search space size and refinement cost of the generalized algorithm, and will provide a qualitative explanation of the effect of various design choices on these factors.

(v) Seen as instantiations of our generalized algorithm, most existing partial order planners differ along the dimensions of the protection strategies they use and the tractability refinements (i.e., refinements whose primary purpose is to reduce refinement cost at the expense of increased search-space size) they employ. Using the qualitative model of design tradeoffs provided by our analysis, we will develop hypotheses regarding the effect of these dimensions of variation on performance. Specifically, we will predict the characteristics of the domains where eager tractability refinements and stronger protection strategies will improve performance. We will validate these predictive hypotheses with the help of a series of focused empirical studies involving a variety of normalized instantiations of our generalized planning algorithm.

Organization: The paper is organized as follows: Section 2 provides the preliminaries of refinement search, develops a model for estimating the size of the search space explored by a refinement search, and introduces the notions of systematicity and strong systematicity. Section 3 reviews the classical planning problem, and provides semantics of plan-space planning in terms of refinement search. Specifically, the notion of a candidate set of a partial plan is formally defined in this section, and the ontology of constraints used in representing partial plans is described. Sections 2 and 3 develop a fair amount of formal machinery and new terminology. Casual readers may want to skim over these sections on the first reading (relying on the glossary and list of symbols in the appendix for reference).

Section 4 describes the generalized refinement planning algorithm, `Refine-Plan`, discusses its various components, and shows how the various ways of instantiating the component steps correspond to the various design choices for partial-order planning. Section 5 shows how the existing plan-space planners, including TWEAK [3], SNLP [24], UA [27] and Nonlin [39] can be seen as instantiations of `Refine-Plan`. It also discusses how `Refine-Plan` can be instantiated to give rise to a variety of new planning algorithms with interesting tradeoffs.

Section 6 develops a model for estimating the search space size and refinement cost of the `Refine-Plan` algorithm, and uses it to develop a qualitative model of the tradeoffs offered by the different design choices (ways of instantiating `Refine-Plan`). Section 7 develops some hypotheses regarding the the effect of various design choices on practical performance. Section 8 reports on a series of focused empirical studies aimed at evaluating these hypotheses. Section 9 discusses the relations between our work and previous efforts on comparing planners. Section 10 summarizes the contributions of the paper

Appendix A provides a quick reference for the list of symbols used in the paper, and Appendix B contains a glossary of terms introduced in the paper.

2 Introduction to Refinement Search

The refinement search (also called split-and-prune search [28]) paradigm is useful for modeling search problems in which it is possible to enumerate all potential solutions (called *candidates*) and verify if one of them is a solution for the problem. The search process can be visualized as a process of starting with the set of *all* potential solutions, and splitting the set repeatedly until a solution can be picked up from one of the sets in *bounded* time. Each search node $\mathcal{N}$ in the refinement search thus corresponds to a set of candidates, denoted by $\langle\langle\mathcal{N}\rangle\rangle$. Figure 1 shows a schematic diagram illustrating the refinement search process (it also illustrates much of the terminology introduced in this section).

A refinement search is specified by providing a set of refinement operators (strategies) **R**, and a solution constructor function `sol`. The search process starts with the initial node $\mathcal{N}_\emptyset$, which corresponds to the set of all candidates (we shall call this set the **candidate space** of the problem, and denote it by $\mathcal{K}$).

The search progresses by generating children nodes by the application of refinement operators. Refinement operators can be seen as set splitting operations on the candidate sets of search nodes. The search terminates when a node $\mathcal{N}$ is found for which the solution constructor returns a solution. The formal definitions of refinement operator and solution constructor follow:

Definition 1 *A refinment operator $\mathcal{R}$ maps a node $\mathcal{N}$ to a set of children nodes $\{\mathcal{N}'_i\}$ such that the candidate sets of each of the children are proper subsets of the candidate set of $\mathcal{N}$ (i.e., $\forall_{\mathcal{N}'_i}\langle\langle\mathcal{N}'_i\rangle\rangle \subset \langle\langle\mathcal{N}\rangle\rangle$).*

$\mathcal{R}$ is said to be **complete** *if every solution belonging to the candidate set of $\mathcal{N}$ belongs to the candidate set of at least one of the children nodes.*

$\mathcal{R}$ is said to be **systematic** *if $\forall_{\mathcal{N}'_i,\mathcal{N}'_j,i\neq j}\ \langle\langle\mathcal{N}'_i\rangle\rangle \cap \langle\langle\mathcal{N}'_j\rangle\rangle = \emptyset$.*

Definition 2 (Solution Constructor) *A solution constructor* `sol` *is a 2-place function which takes a search node $\mathcal{N}$ and a solution criterion $\mathcal{S}_\mathcal{G}$ as arguments. It will return either one of three values:*

(i) **fail*, meaning that no candidate in $\langle\langle\mathcal{N}\rangle\rangle$ satisfies the solution criterion* *(ii) Some candidate $k \in \langle\langle\mathcal{N}\rangle\rangle$ which satisfies the solution criterion (i.e., k is a solution)*

(iii) $\perp$, meaning that `sol` *can neither return a solution, nor determine that no such candidate exists.*

In the first case, $\mathcal{N}$ can be pruned. In the second case, search terminates with success, and in the third, $\mathcal{N}$ will be refined further. $\mathcal{N}$ is called a **solution node** *if the call* $\texttt{sol}(\mathcal{N}, \mathcal{S}_{\mathcal{G}})$ *returns a solution.* [4]

Definition 3 (Completeness of Refinement Search) *A refinement search with the refinement operator set* **R** *and a solution constructor function* `sol` *is said to be complete if for every solution k of the problem, there exists some search node $\mathcal{N}$ that results from a finite number of successive refinement operations on $\mathcal{N}_\emptyset$, (i.e., $\exists_{\mathcal{R}_i \in \mathbf{R}} \mathcal{N} = \mathcal{R}_1(\mathcal{R}_2 \cdots (\mathcal{R}_n(\mathcal{N}_\emptyset)))$, where $\mathcal{N}_\emptyset$ is the node whose candidate set is the entire candidate space $\mathcal{K}$), such that* `sol` *can pick up k from $\mathcal{N}$.*

Notice that the completeness of search depends not only on the refinement strategies, but also on the *match between the solution constructor function and the refinement strategies*. It can be shown easily that for finite candidate spaces, and solution constructors that are powerful enough to pick solutions from singleton sets in bounded time, completeness of refinement operators is *sufficient* to guarantee the completeness of refinement search. [5]

Search nodes as Constraint Sets: Although it is conceptually simple to think of search nodes in terms of their candidate sets, we obviously do not want to represent the candidate sets explicitly in our implementations. Instead, the candidate sets are typically implicitly represented as generalized constraint sets associated with search nodes (c.f. [10]) such that every candidate that is *consistent* with the constraints in that constraint set is taken to belong to the candidate set of the search node. Under this representation, the refinement of a search node corresponds to adding new constraints to its constraint set, thereby restricting its candidate set.

Any time the set of constraints of a search node becomes inconsistent (unsatisfiable), the candidate set becomes empty. Since there is no utility in refining an empty candidate set, such inconsistent nodes can be pruned, optionally, from the search space. When such pruning is done, it can reduce the over all size of the search tree. However, depending upon the type of the constraints, verifying that a node is inconsistent can be very costly. Thus, the optional pruning step trades the cost

[4] It is instructive to note that a solution-constructor may return *fail* even if the candidate set of the node is not empty. The special case of nodes with empty candidate sets is usually handled by consistency check, see below.

[5] This condition is not necessary because the individual refinements need not be complete according to the strong definition of Definition 1 -- specifically, it is enough if the refinements never lose a minimal solution.

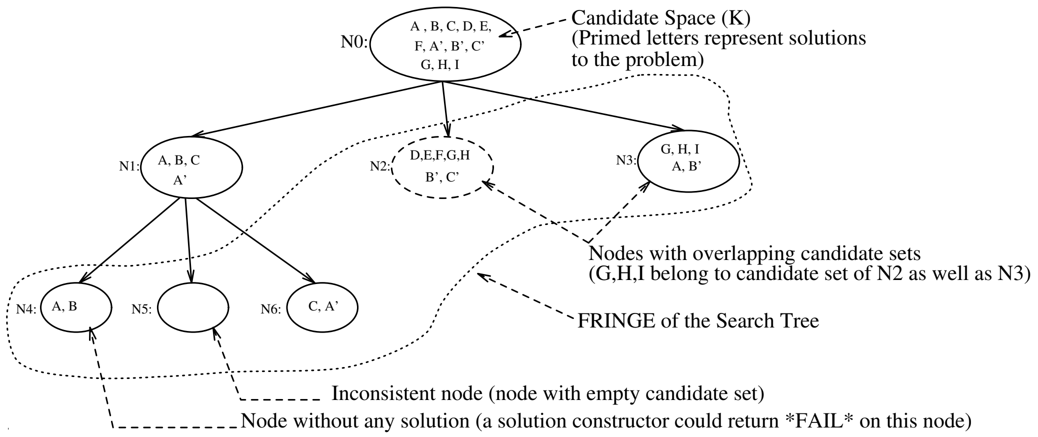


Fig. 1. Schematic diagram illustrating refinement search. Here, the candidate space ($\mathcal{K}$) is the set $\{A, B, C, D, E, F, G, H, I, A', B', C'\}$. The fringe ($\mathcal{F}$) is given by the set $\{N2, N3, N4, N5, N6\}$. The average size of the candidate sets of the nodes on the fringe, κ_d, is $\frac{16}{5}$, and the redundancy factor ρ for the fringe is $\frac{16}{12}$. It is easy to verify that $|\mathcal{F}_d| = \frac{|\mathcal{K}| \times \rho_d}{\kappa_d}$.

Algorithm Refine-Node($\mathcal{N}$) **Parameters**: (i) `sol`: Solution constructor function. (ii) **R**: Refinement operators.

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

1. Refinements: Pick a refinement operator $\mathcal{R} \in$ **R**, (*Not a backtrack point.*)

Nondeterministically choose a refinement $\mathcal{N}'$ from $\mathcal{R}(\mathcal{N})$ (the refinements of n with respect to $\mathcal{R}$.)

(*Note*: It is legal to repeat this step multiple times per invocation)

2. Consistency Check: (*Optional*) If $\mathcal{N}'$ is inconsistent, fail. Else, continue.

3. Recursive Invocation: Recursively invoke `Refine-Node` on $\mathcal{N}'$.

Fig. 2. A recursive non-deterministic algorithm for generic refinement search. The search is initiated by invoking Refine-Node($\mathcal{N}_\emptyset$)

of consistency check against the reduction in the search space afforded through pruning.

Definition 4 (Inconsistent Search Nodes) *A search node is said to be inconsistent if its candidate set is empty, or equivalently, its constraint set is unsatisfiable.*

Definition 5 (Informedness) *A refinement search is said to be informed if it never refines an inconsistent search node.*

Search Space Size: Figure 2 outlines the general refinement search algorithm. To characterize the size of the search space explored by this algorithm, we will look at the size of the fringe (number of leaf nodes) of the search tree. Suppose $\mathcal{F}_d$ is the d^{th} level fringe of the search tree explored by the refinement search (in a breadth-first search). Let $\kappa_d \geq 0$ be the average size of the candidate sets of the search nodes in the d^{th} level fringe, and $\rho_d (\geq 1)$ be the redundancy factor, i.e., the average number of search nodes on the fringe whose candidate sets contain a given candidate in $\mathcal{K}$. 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). If b is the average branching factor of the search, then the size of d^{th} level fringe is also given by $O(b^d)$. Thus, we have,

$$|\mathcal{F}_d| = \frac{|\mathcal{K}| \times \rho_d}{\kappa_d} = O(b^d) \tag{1}$$

In terms of this model, a minimal guarantee one would like to provide is that the size of the fringe will never be more than the size of the overall candidate space $|\mathcal{K}|$. Trying to ensure this motivates two important notions of irredundancy in refinement search: *systematicity* and *strong systematicity*.

Definition 6 (Systematicity) *A refinement search is said to be* **systematic** *if for any two nodes $\mathcal{N}$ and $\mathcal{N}'$ falling in different branches of the search tree, then $\langle\langle\mathcal{N}\rangle\rangle \cap \langle\langle\mathcal{N}'\rangle\rangle = \emptyset$ (i.e., the candidate sets represented by $\mathcal{N}$ and $\mathcal{N}'$ are disjoint).*

Definition 7 (Strong Systematicity) *A refinement search is said to be strongly systematic if it is both systematic and informed (see Definition 5).*

From the above, it follows that for a systematic search, the redundancy factor, ρ, is 1. Thus, the sum of the cardinalities of the candidate sets of the termination fringe will be no larger than the set of all candidates $\mathcal{K}$. For strongly systematic search, in addition to ρ being equal to 1, we also have $\kappa_d \geq 1$ (since no node has an empty candidate set) and thus $|\mathcal{F}_d| \leq |\mathcal{K}|$. Thus,

Proposition 8 *The fringe size of any search tree generated by a strongly systematic refinement search is strictly bounded by the size of the candidate space (i.e. $|\mathcal{K}|$).*

It is easy to see that a refinement search is systematic if all the individual refinement operations are systematic. To convert a systematic search into a strongly systematic one, we only need to ensure that all inconsistent nodes are pruned from the search. The complexity of the consistency check required to effect this pruning depends upon the nature of the constraint sets associated with the search nodes.

3 Planning as Refinement Search

3.1 Informal Overview

Given a planning problem, plan-space planners attempt to solve it by searching in the space of ''partial plans.'' The partial plans are informally understood as incomplete solutions. The search process starts with an empty plan, and successively adds ''details'' (steps, orderings, etc.) to it until it becomes a correct plan for solving the problem. Without attaching a formal meaning to partial plans, it is hard to explain the semantic implications of this process.

In this section, we will provide semantics for partial plans in terms of refinement search. In this view, partial plans are seen not as incomplete solutions, but as representations for *sets* of potential solutions (candidates). Planning is seen as the process of splitting these candidate sets until a solution is found. In the subsequent sections, we shall show that this view provides a powerful unifying framework.

To provide a formal account of this process, we need to define the notion of the candidate set of a partial plan, and we tie this semantic notion to some syntactic characteristic of the partial plan. We start by noting that the solution for a planning problem is ultimately a sequence of operators (actions), which when executed from an initial state, results in a state that satisfies all the goals of the problem. Thus, ground operator sequences constitute potential solutions for any planning problem, and we will define the candidate set of a partial plan as all the ground operator sequences that are consistent with all the constraints in the partial plan. Accordingly, the steps, orderings and bindings of the partial plan are seen as imposing constraints on which ground operator sequences do and do not belong to the candidate set of the plan. The empty plan corresponds to all the ground operator sequences since it doesn't impose any constraints.

For example, consider the scenario of solving a blocks world problem of moving three blocks A, B and C from the table to the configuration $On(A, B) \wedge On(B, C)$. Suppose at some point during the search we have the following partial plan:

$$\mathcal{P}_B : \texttt{Start} - Move(A, Table, B) - \texttt{Fin}$$

We will see it as a stand-in for all ground operator sequences which contain the operator instance $Move(A, Table, B)$ in it. In other words, the presence of the step $Move(A, Table, B)$ eliminates from the candidate set of the plan any ground operator sequence that *does not contain* the action $Move(A, Table, B)$. Operator sequences such as $Move(A, Table, C) - Move(B, Table, A) - Move(A, Table, B)$ *are* candidates of the partial plan $\mathcal{P}_B$.

One technical problem with viewing planning as a refinement search, brought out by the example above, is that the candidate sets of partial plans are potentially infinite. In fact, the usual types of constraints used by plan-space planners are such that no partial plan at any level of refinement in the search tree will have a ''singleton candidate sets.'' [6] This means that the usual mental picture of refinement search as the process of ''splitting sets until they become singletons'' (see Section 2) is not valid. In addition, tractable solution constructor functions cannot hope to look at the full candidate sets of partial plans at *any level of refinement*.

To handle this problem, the solution constructor functions in planning look at only the ''minimal candidates'' of the plan. Intuitively, minimal candidates are ground operator sequences that will not remain candidates if any of the operators are removed from them. In the example plan $\mathcal{P}_B$ described earlier, the only minimal candidate is $Move(A, Table, B)$. All other candidates of a partial plan can be derived by starting from a minimal candidate and adding operators without violating any plan constraints. As the refinements continue, the minimal candidates of a partial plan increase in length, and the solution constructors examine to see if one of them is a solution. This can be done in bounded time since the set of minimal candidates of a partial plan is finite (this is because an n-step plan has at most $n!$ linearizations). Figure 3 illustrates this view of the candidate set of a partial plan.

Finally, to connect this view to the syntactic operations performed by planners, we need to provide a relation between the candidate set of the plan and some syntactic notion related to the plan. We do this by fixing a one-to-one correspondence between minimal candidates (a semantic concept) and a syntactic notion called the safe ground linearizations of the plan.

In the rest of this section, we formalize these informal ideas. We will start by reviewing the notion of solution of a planning problem (Section 3.2). Next, we provide a syntactic description of the constraints comprising a partial plan (Section 3.3). At this point we will develop the syntactic and semantic notions of satisfying the constraints of the partial plan. The semantic notion depends on the concept of ground operator sequences, while the syntactic notion depends on the idea of ground linearizations.

We will then provide semantics of partial plans in terms of their candidate sets, which are ground operator sequences satisfying all the constraints of the partial plan, and show that executable candidates of the plan correspond to solutions to the planning problem (Section 3.4). Finally, we will relate the semantic notion of the candidate set of a partial plan to a syntactic notion called safe ground linearization of the partial plan (see Figure 3 and Section 3.5). Specifically we will show that the safe ground linearizations correspond to the minimal candidates of a partial plan

[6] For a partial plan to have a singleton candidate set, the constraints on the plan must explicitly disallow addition of new operators to the plan. The ''immediacy'' constraints, discussed by Ginsberg in [9] are an example of such constraints (see Section 9).

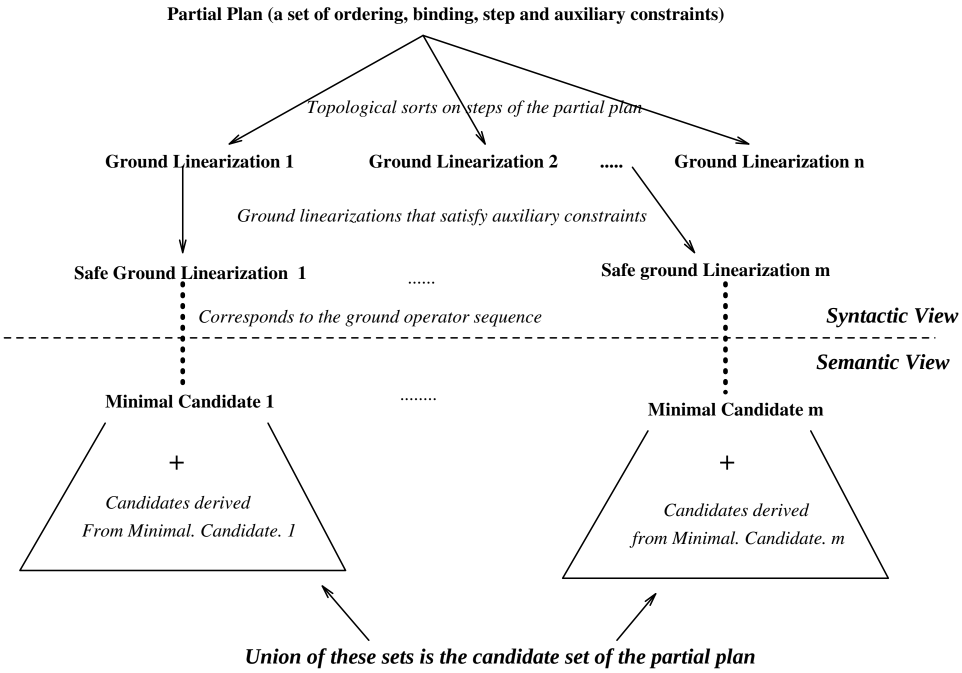


Fig. 3. A schematic illustration of the relation between a partial plan and its candidate set. The candidate set of a partial plan is all ground operator sequences that are consistent with its constraints. These can be seen in terms of minimal candidates (which correspond to the safe ground linearizations of the partial plan) and ground operator sequences derived from them by adding more operators.

(i.e., the smallest length ground operator sequences belonging to the candidate set of a plan). This allows us to provide meanings to syntactic operations on the partial plan representation in terms of their import on the candidate set of the partial plan.

3.2 Solutions to a planning problem

Whatever the exact nature of the planner, the ultimate aim of (classical) planning is to find a sequence of *ground operators*, which when executed in the given initial state, will produce desired *behaviors* or sequences of world states. Most classical planning techniques have traditionally concentrated on the attainment of goals[8]. These goals can be seen as a sub-class of behavioral constraints, which restricts the agent's attention to behaviors that end in world states satisfying desired properties. For the most part, this is the class of goals we shall also be considering in this paper. [7] Below, we assume that a planning problem is a pair of world states, $[\mathcal{I}, \mathcal{G}]$, where $\mathcal{I}$ is the initial state of the world, and $\mathcal{G}$ is the specification of the desired behaviors.

[7] In [15,17] we show that our framework can be easily extended to deal with to a richer class of behavioral constraints, including maintenance goals and intermediate goals.

The operators (also called actions) in classical planning are modeled as general *state transformation functions*. Pednault [30] provides a logical representation, called the *Action Description Language* (ADL) for representing such state transformation functions. 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 precondition formulas can also have disjunction, but disjunction is not allowed in the effects formula. The subset of this representation where both formulas can be represented as conjunctions of function-less first order *literals*, and all the variables have infinite domains, is called the TWEAK representation (c.f. [3,14,42]).[8] A ground operator is an operator that does not contain any uninstantiated variables.

Given a set of ground operators, we can form a space of ground operator sequences, only a subset of which forms solutions to a planning problem. For any planning problem the space of **all** ground operator sequences is called the *candidate space* of that problem. As an example of this space, if a domain contains three ground operators $a1$, $a2$ and $a3$, then the candidate space of any problem would be a subset of the regular expression $\{a1|a2|a3\}^*$.

We now formally define the semantic meaning of a solution to a planning problem.

Definition 9 (Solution of a Planning Problem) *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 specification of the desired behaviors, if the following two restrictions are satisfied:*

(i) S is executable, *i.e.,* $\mathcal{I} \vdash prec(o_1)$, $o_1(\mathcal{I}) \vdash prec(o_2)$ *and* $o_{n-1}(o_{n-2} \cdots (o_1(\mathcal{I}))) \vdash prec(o_n)$ *(where $prec(o)$ denotes the precondition formula of the operator o) and*
(ii) The sequence of states $\mathcal{I}$, $o_1(\mathcal{I})$, $\cdots$, $o_n(o_{n-1} \cdots (o_1(\mathcal{I})))$ satisfies the behavioral constraints specified in the goals of the planning problem.

For goals of attainment, the second requirement is stated solely in terms of the last state resulting from the plan execution: $o_n(o_{n-1} \cdots (o_1(\mathcal{I}))) \vdash \mathcal{G}$. A solution S is said to be **minimal** *if no operator sequence obtained by removing some of the operators from S is also a solution.*

3.3 Syntactic Definition of Partial-Order Plans

[8] In TWEAK representation, the list of non-negated effects is called the *Add* list while the list of negated effects is called the *Delete* list.

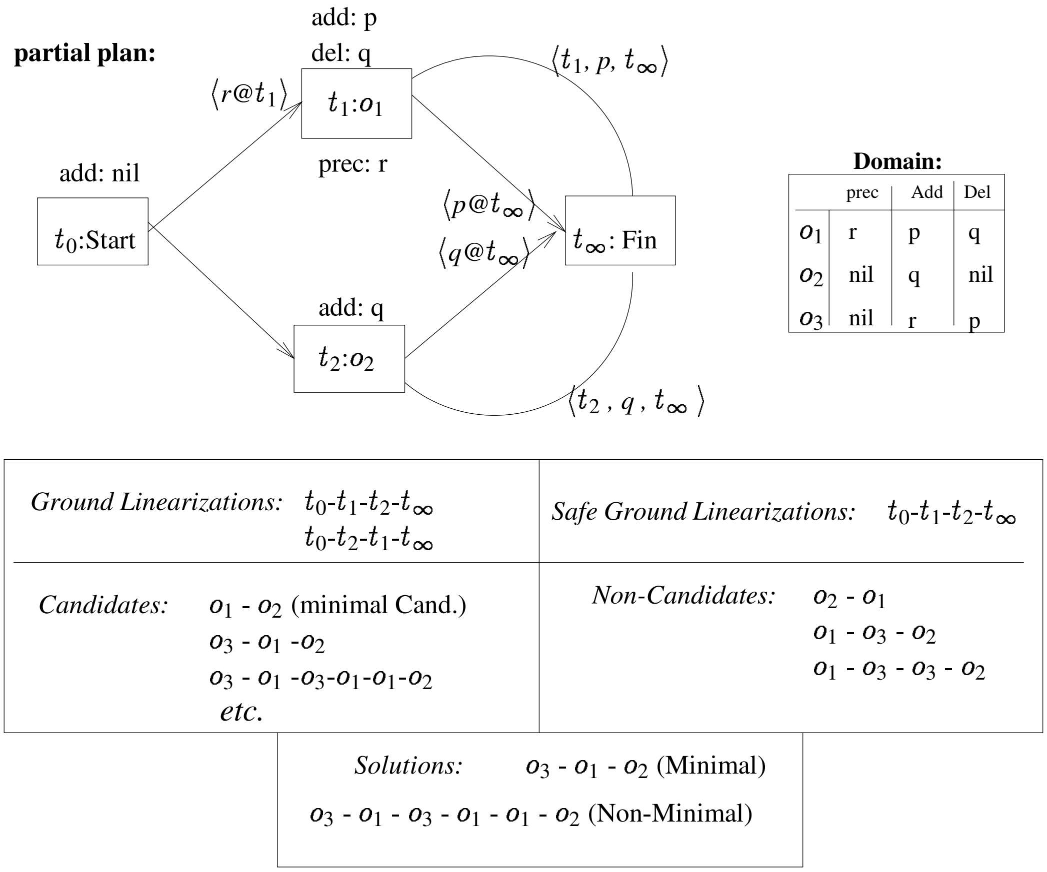


Fig. 4. An example partial plan illustrating the terminology used in describing the candidate sets. The table on top right shows the preconditions and effects of the operators. The effects of `start` operator, correspond to the initial state of the problem while the preconditions of `Fin` correspond to the top level goals of the plan. In this example, initial state is assumed to be null, and the top level goals are assumed to be p and q.

Formally, a partial plan is a 5-tuple: $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ where:

-- T is the set of steps in the plan. Elements of T are denoted by symbols s, t and their subscripted versions. T contains two distinguished steps t_0 and t_∞.
-- $\mathcal{ST}$ is a symbol table, which maps steps to ground operators in the domain.
-- O is a partial ordering relation over T that specifies the constraints on the order of execution of the plan steps. By definition, O orders t_0 to precede all other steps of the plan, and t_∞ to follow all others steps of the plan.
-- $\mathcal{B}$ 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).

In a symbol table $\mathcal{ST}$, the special step t_0 is always mapped to the dummy operator `start`, and similarly t_∞ is always mapped to the dummy operator `fin`. The effects of `start` and the preconditions of `fin` correspond, respectively, to the initial state and the desired goals (of attainment) of the planning problem. The symbol table $\mathcal{ST}$, together with T provides a way of distinguishing between multiple occurrences of the same operator in the given plan.

The auxiliary constraints $\mathcal{L}$ deserves more explanation. In principle, these include any constraints on the partial plan that are not classifiable into ''steps'', ''orderings'' and ''bindings''. Two important types of auxiliary constraints we will discuss in detail later, are *Interval preservation constraints* (IPCs), and *Point truth constraints* (PTCs). An IPC is represented as a 3-tuple $\langle s, p, t\rangle$, while a PTC is represented as a 2-tuple $\langle p@t\rangle$. Informally, the IPC $\langle s, p, t\rangle$ requires that the condition p be preserved between the steps s and t, while the PTC $\langle p@t\rangle$ requires that the condition p be true before the step t. IPCs are used to represent bookkeeping (protection) constraints (Section 4.3) while PTCs are used to capture the pre-requisites that need to be true before each of the steps in the plan. In particular, given any partial plan $\mathcal{P}$, *corresponding to every precondition C of step s in the plan, the partial plan contains a PTC (non-monotonic auxiliary constraint)* $\langle C@s\rangle$. Auxiliary constraints can also be used to model other aspects of refinement planning. In [15], we show that IPC's can be used to model maintenance goals while PTC's can be used to model filter conditions.

Example: Figure 4 shows an example partial plan $\mathcal{P}_E$, whose constraint set appears below:

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

$\mathcal{P}_E$ contains four steps t_0, t_1, t_2, t_∞. These steps are mapped to the *syntactic operators* in this domain, `Start`, o_1, o_2, `fin`, respectively. The preconditions and effects of these operators are described in the table at the top right corner of Figure 4. The orderings between the steps are shown by arrows. The plan also contains a set of five auxiliary constraints -- 2 IPCs and 3 PTCs.

3.3.1 Ground Linearizations of a Partial Plan

Definition 10 (Ground Linearizations) *A* **ground linearization** *(also called 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 $\mathcal{B}$.*

Example: In our example partial plan $\mathcal{P}_E$, there are two ground linearizations $t_0t_1t_2t_\infty$ and $t_0t_2t_1t_\infty$.

A ground linearization captures the syntactic notion of what it means for a partial plan to be consistent with its own ordering and binding constraints. If a plan has no ground linearizations, then it means that it is not consistent with its ordering and binding constraints. We can extend this notion to handle the auxiliary constraints as follows:

Definition 11 *A ground linearization G of a partial plan $\mathcal{P}$ is said to satisfy an IPC $\langle t, p, t' \rangle$ of $\mathcal{P}$, if for every step t'' between t and t' in G, the operator $\mathcal{ST}[t'']$ does not delete p.*

Definition 12 *G is said to satisfy a PTC $\langle c@t \rangle$ if there exists a step t' before t in G (t' could be t_0), such that $\mathcal{ST}(t')$ has an effect c and for every step t'' between t' and t in G, $\mathcal{ST}(t')$ does not delete c.*

Definition 13 *A partial plan $\mathcal{P}$ is said to be consistent with an auxiliary constraint if at least one of its ground linearizations satisfies it.*

Example: Consider the ground linearization $G_1 : t_0t_1t_2t_\infty$ of our example plan $\mathcal{P}_E$. G_1 satisfies the IPC $\langle t_1, p, t_\infty \rangle$ since the operator o_2 corresponding to t_2, which comes between t_1 and t_∞ in G_1, does not delete p. Thus $\mathcal{P}_E$ itself is consistent with the IPC $\langle t_1, p, t_\infty \rangle$. Similarly, G_1 also satisfies the PTC $\langle q@t_\infty \rangle$ since t_2 gives q as an effect and there is no step between t_2 and t_∞ deleting q in G_1.

3.4 Candidate Set Semantics of a Partial Plan

Having defined the syntactic representation of a partial plan, we need to answer the question -- what does it represent? In this section, we provide formal semantics for partial plans based on the notion of candidate sets. Among other things, we explain what it means to add additional syntactic constraints to a partial plan, and what it means for a partial plan to represent a solution to a planning problem.

3.4.1 Mapping Function $\mathcal{M}$

Our interpretation of a partial plan $\mathcal{P}$ corresponds to the set of all ground operator sequences that are consistent with the constraints of $\mathcal{P}$. This correspondence is defined formally via a mapping function $\mathcal{M}$:

Definition 14 (Mapping function) *$\mathcal{M}$ is said to be a mapping function from a plan $\mathcal{P}$ to a ground operator sequence S if*

(i) $\mathcal{M}$ maps all steps of $\mathcal{P}$ (except the dummy steps t_0 and t_∞) to elements of S, such that no two steps are mapped to the same element of S.
(ii) $\mathcal{M}$ agrees with $\mathcal{ST}$ (i.e., $S[\mathcal{M}(t)] = \mathcal{ST}(t)$)
(iii) for any two steps t_i, t_j in $\mathcal{P}$ such that $t_i \prec t_j$, if $\mathcal{M}(t_i) = S[l]$ and $\mathcal{M}(t_j) = S[m]$ then $l < m$.

Example: Considering our example plan $\mathcal{P}_E$ in Figure 4, $\mathcal{M} = \{t_1 \rightarrow S[5], t_2 \rightarrow S[6]\}$ is a mapping function from $\mathcal{P}_E$ to the operator sequence $S : o_3 o_1 o_3 o_1 o_1 o_2$. This is because:

(i) $S[5]$ is o_1 which is also $\mathcal{ST}(t_1)$, and similarly $S[6]$ is o_2 which is also $\mathcal{ST}(t_2)$.
(ii) There are no ordering relations between t_1 and t_2 in $\mathcal{P}_E$ and thus S trivially satisfies the ordering relations of $\mathcal{P}_E$.

3.4.2 Auxiliary Constraints

Intuitively, the last two clauses of the definition of the mapping function ensure that the ground operator sequence satisfies the steps and orderings of the partial plan. We could also define what it means to say that a ground operator sequence satisfies an auxiliary constraint.

Formally, an IPC $\langle t_i, c, t_j \rangle$ of a plan $\mathcal{P}$ is said to be satisfied by a ground operator sequence S, under a mapping function $\mathcal{M}$, if and only if every operator o in S between $\mathcal{M}(t_i)$ and $\mathcal{M}(t_j)$ preserves (does not delete) the condition c. For readers who are familiar with the ''causal link'' notation [24], note that an IPC $\langle s_i, c, s_j \rangle$ *does not* require that s_i give the condition c, but merely that the condition c be preserved (i.e., left unaffected) in the interval between s_i and s_j.

The IPCs are examples of **monotonic** auxiliary constraints. An auxiliary constraint $\mathcal{C}$ is monotonic if for any ground operator sequence S that does not satisfy C under $\mathcal{M}$, adding operators to S will not make it satisfy C either. A constraint that is not monotonic is called **non-monotonic**.

Example: In our example plan $\mathcal{P}_E$, the operator sequence $S : o_3 o_1 o_3 o_1 o_1 o_2$ will satisfy the IPCs with respect to the mapping function $\mathcal{M} = \{t_1 \rightarrow S[5], t_2 \rightarrow S[6]\}$. In particular, the IPC $\langle t_1, p, t_\infty \rangle$ is satisfied because all the operators between $S[5]$ and the end of the operator sequence, which in this case is just o_2, preserve (do not violate) p. This can be verified from the effects of the operators described in the top right table in Figure 4. It can also be verified that the IPC would not be

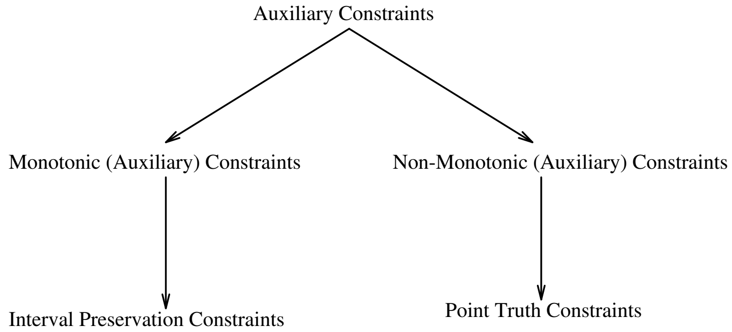


Fig. 5. The relation between the various types of auxiliary constraints

satisfied with respect to a different mapping, $\mathcal{M}' = \{t_1 \to S[2], t_2 \to S[6]\}$ (since $S[3] = o_3$ deletes p).

Similarly, a point truth constraint [9] $\langle c@t \rangle$ is said to be satisfied by a ground operator sequence S under $\mathcal{M}$, if and only if

(i) either c is true in the initial state, and is preserved by every action of S occurring before $\mathcal{M}(t)$, or
(ii) c is made true by some action $S[j]$ that occurs before $\mathcal{M}(t)$, and is preserved by all the actions between $S[j]$ and $\mathcal{M}(t)$.

Example: Consider once again the example plan $\mathcal{P}_E$, the operator sequence $S : o_3 o_1 o_3 o_1 o_1 o_2$ and the mapping function $\mathcal{M} = \{t_1 \to S[5], t_2 \to S[6]\}$. The PTC $\langle r@t_1 \rangle$ is satisfied by S with respect to $\mathcal{M}$ since $S[3] = o_3$ adds r and $S[4] = o_1$ does not delete r. It can be verified that the other two PTCs are also satisfied by S. This is because $S[5] = o_1$ gives p and $S[6] = o_2$ gives q without deleting p, and thus both p and q are true at the end S.

The PTCs are examples of *non-monotonic* auxiliary constraints. To see this, consider the operator sequence $S : o_1 o_2$. S fails to satisfy the PTC $\langle r@t_1 \rangle$ of $\mathcal{P}_E$ (Figure 4) with respect to the mapping $\mathcal{M} = \{t_1 \to S[1], t_2 \to S[2]\}$. However $S' : o_3 o_1 o_2$ will satisfy the PTC with respect to the same mapping.

Figure 5 shows the relationship between the different types of auxiliary constraints that we have defined above.

3.4.3 *Candidate Set of a Partial Plan*

Now that we have defined what it means for a ground operator sequence to satisfy the step, ordering, binding and auxiliary constraints, it is time to formally define when a ground operator sequence becomes a candidate. Intuitively, it would seem

[9] This is called a *point-protected condition* in [41].

reasonable to go ahead and say that a candidate is a ground operator sequence that satisfies all the constraints of the plan with respect to the same mapping. This however leads to a technical difficulty.

A useful property that we want for a candidate is that given a ground operator sequence S that is not a candidate of a partial plan, adding operators to S should not make it a candidate.[10] For this to happen, we want the auxiliary constraints to be such that given an operator sequence S that does not satisfy an auxiliary constraint $\mathcal{C}$ with respect to a mapping $\mathcal{M}$, adding more operators to S will not make it satisfy $\mathcal{C}$. From our previous discussion, we note that monotonic auxiliary constraints, which include IPCs, have this property, while non-monotonic auxiliary constraints don't. Accordingly, we define the candidate set of a partial plan in terms of its monotonic auxiliary constraints.

Definition 15 (Candidate set of a Partial plan) *Given a partial plan $\mathcal{P}$: $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$, a ground operator sequence S is said to be a candidate of $\mathcal{P}$ if there is a mapping function $\mathcal{M}$ from $\mathcal{P}$ to S with respect to which S satisfies all the monotonic auxiliary constraints of $\mathcal{P}$.*

The candidate set of a partial plan is the set of all ground operator sequences that are its candidates.

An operator sequence S is a **minimal candidate***, if it is a candidate, and no operator sequence obtained by removing some of the operators from S is also a candidate of $\mathcal{P}$.*

Note that by our definition, a candidate of a partial plan might not be executable. It is possible to define candidate sets only in terms of executable operator sequences (or ground behaviors), but we will stick with this more general notion of candidates since generating an executable operator sequence can itself be seen as part of planning activity.

Definition 16 (Solution of a Partial Plan) *A ground operator sequence S is said to be a* solution of a partial plan *$\mathcal{P}$, if S is executable and S is a candidate of $\mathcal{P}$ with respect to a mapping $\mathcal{M}$, and S satisfies all the non-monotonic auxiliary constraints of $\mathcal{P}$ with respect to $\mathcal{M}$.*

It can be verified that for minimal candidates, executability is automatically

[10] To understand the motivation behind this, recall, from Figure 3, that we want to define candidate sets in such a way that planners can concentrate on the minimal candidates (which ''correspond'' to the safe ground linearizations of the partial plan). Accordingly, we would like to ensure that if none of the ground operator sequences corresponding to the ground linearizations of a partial plan satisfy the auxiliary constraints, the plan will have an empty candidate set (so that we could go ahead and prune such plans without losing completeness).

guaranteed if all the non-monotonic auxiliary constraints are satisfied (recall that corresponding to each precondition c of each step s of the plan, the partial plan contains a PTC $\langle c@s \rangle$). Finally, it can also be verified that the solutions of a partial plan correspond to the solutions of the planning problem $[\mathcal{I}, \mathcal{G}]$, where $\mathcal{I}$ is the effect formula of t_0, and $\mathcal{G}$ is the precondition formula of t_∞ of $\mathcal{P}$ according to Defn. 9.

Example: Continuing with our example plan $\mathcal{P}_E$, the operator sequence $S : o_3 o_1 o_3 o_1 o_1 o_2$ and the mapping function $\mathcal{M} = \{t_1 \rightarrow S[5], t_2 \rightarrow S[6]\}$, we have already seen that S satisfies the step, ordering and interval preservation constraints with respect to the mapping $\mathcal{M}$. Thus S is a candidate of the partial plan $\mathcal{P}_E$. S is however not a minimal candidate since the sequence $S' : o_1 o_2$ is also a candidate of $\mathcal{P}_E$ (with the mapping function $\mathcal{M} = \{t_1 \rightarrow S[1], t_2 \rightarrow S[2]\}$), and S' can be obtained by removing elements from S. It can be easily verified that S' is a minimal candidate.

Since S also satisfies the PTCs with respect to this mapping, and since S is executable, S is also a solution of $\mathcal{P}_E$. S is however not a minimal solution, since it can be verified that $S'' : o_3 o_1 o_2$ is also a solution, and S'' can be derived by removing elements from S. It is interesting to note that although S'' is a minimal solution, it is *not* a minimal candidate. This is because, as discussed above, $S' : o_1 o_2$ is a candidate of $\mathcal{P}_E$ (note that S' is not a solution). This example illustrates the relations between the candidates, minimal candidates, solutions and minimal solutions.

3.4.4 Summarizing the Meaning of Partial Plans in terms of Candidate Sets

We are now in a position to summarize the refinement-search based semantics of partial plans. A partial plan $\mathcal{P}$ can be equivalently understood as its candidate set $\langle\langle \mathcal{P} \rangle\rangle$. A subset of $\langle\langle \mathcal{P} \rangle\rangle$, called the solutions of $\mathcal{P}$, corresponds to the actual solutions to the planning problem. The process of finding these solutions, through refinement search, can be seen as splitting the candidate sets of the plan in such a way that the minimal candidates of the resulting partial plans correspond to solutions of the planning problem.

The main refinement operation used to achieve this involves the so called establishment refinement (Section 4.2). In essence this can be understood in terms of taking a PTC of a partial plan, and splitting the candidate set of the partial plan such that the minimal candidates of the each resulting child plan will satisfy that PTC. After working on each PTC this way, the planner will eventually reach a partial plan one of whose minimal candidates satisfies *all* the PTCs. At this point, the search can terminate with success (recall that the partial plan contains a PTC corresponding to every precondition of every step of the plan).

3.5 Relating Candidate sets and Ground Linearizations

In the previous section, we defined the candidate-set semantics of partial plans. Candidate-set semantics can be thought of as providing a denotational semantics for partial plans in refinement planning. However, actual refinement planners do not deal with candidate sets explicitly during planning, and instead make some syntactic operations on the partial plans. To understand the semantic import of these operations, we need to provide a correspondence between the candidate set of a partial plan and some syntactic notion related to the partial plan.

In particular, we define the notion of *safe* ground linearizations.

Definition 17 (Safe Ground Linearization) *A ground linearization G of a plan $\mathcal{P}$ is said to be* **safe**, *if it satisfies all the monotonic auxiliary constraints (IPCs for our representation).*

Example: Consider the ground linearization $G_1 : t_0 t_1 t_2 t_\infty$ of our example plan $\mathcal{P}_E$. G_1 satisfies the IPC $\langle t_1, p, t_\infty \rangle$ since the operator o_2 corresponding to t_2, which comes between t_1 and t_∞ in G_1, does not delete p. Similarly, we can see that the IPC $\langle t_2, q, t_\infty \rangle$ is also satisfied by G_1. Thus, G_1 is a safe ground linearization. In contrast, the other ground linearization $G_2 : t_0 t_2 t_1 t_\infty$ is not a safe ground linearization since the IPC $\langle t_2, q, t_\infty \rangle$ is not satisfied (t_1 which comes between t_1 and t_∞ in G, corresponds to the operator o_1 which delete q).

We will now put the candidate set of a plan in correspondence with the safe ground linearization. To do this, we first define what it means for an operator sequence to correspond to a ground linearization of a plan.

Definition 18 *Let G be a ground linearization of a plan $\mathcal{P}$. Let G' be the sequence derived by removing t_0 and t_∞ from G. An operator sequence S is said to* **correspond** *to the ground linearization G, if $\forall_i S[i] = \mathcal{ST}(G'[i])$ (where $S[i]$ and $G'[i]$ are the i^{th} elements of S and G' respectively)*

Example: In our example partial plan $\mathcal{P}_E$, the ground operator sequence $S_1 : o_1 o_2$ corresponds to the ground linearization $G_1 : t_0 t_1 t_2 t_\infty$ (since $\mathcal{ST}$ maps t_1 to o_1 and t_2 to o_2). Similarly, the ground operator sequence $S_2 : o_2 o_1$ corresponds to the ground linearization $G_2 : t_0 t_2 t_1 t_\infty$.

Proposition 19 (Correspondence Theorem:) *A ground operator sequence S is a minimal candidate of a partial plan $\mathcal{P}$ if and only if it corresponds to some safe ground linearization G of the plan $\mathcal{P}$*

Proof: (If): Let G be a safe ground linearization of $\mathcal{P}$, and G' be the sequence obtained by stripping t_0 and t_∞ from G. Let S be the operator sequence obtained by translating step names in G to operators (via the symbol table $\mathcal{ST}$, such that $S[i] = \mathcal{ST}(G'[i])$. By construction, S corresponds to G. Consider the mapping $\mathcal{M} = \{G'[i] \rightarrow S[i] | \forall_i\}$. It is easy to see that $\mathcal{M}$ is a mapping function from $\mathcal{P}$ to S by Defn. 14. We can also verify that S satisfies all monotonic auxiliary constraints of $\mathcal{P}$ according to $\mathcal{M}$. To see this, consider an IPC $\langle t', p, t\rangle$ of $\mathcal{P}$. Since G is safe, it satisfies the IPC. This means that if $G[i] = t'$ and $G[j] = t$, then all the elments $G[i+1] \cdots G[j-1]$ will preserve p. By the construction of S from G, we know that $S[i]$ will correspond to t' and $S[j]$ will correspond to t under mapping $\mathcal{M}$. Further, it also means that the operators $S[i+1] \cdots S[j-1]$ will preserve p. This means S satisfies the IPC with respect to $\mathcal{M}$.

The above proves that S is a candidate of $\mathcal{P}$ with respect to the mapping function $\mathcal{M}$. In addition, since by construction S corresponds to G, removing any operator from S would leave more steps in G than there are elements in S. This makes it impossible to construct a mapping function from $\mathcal{P}$ to S (since a mapping function must map every step of $\mathcal{P}$ must be mapped to a different element of S). Thus, S is also a minimal candidate.

(Only If): Before we prove the only if part of the correspondence theorem, We will state and prove a useful lemma:

Lemma: A minimal candidate S of a partial plan $\mathcal{P}$ will have exactly as many elements as the number of steps in $\mathcal{P}$ (not counting t_0 and t_∞).

Proof of the Lemma: Let m be the number of steps in $\mathcal{P}$ (not counting t_0 and t_∞. S cannot have less than m elements since if it does then it will be impossible to construct a mapping function from $\mathcal{P}$ to S (recall that a mapping function must map each step to a different element of S). S cannot have greater than m elements, since if it did then S will not be a minimal candidate. To see this, suppose S has greater than m steps, and it is a candidate of $\mathcal{P}$ under the mapping function $\mathcal{M}$. Consider the operator sequence S' obtained by removing from S all the elements which do not have any step of $\mathcal{P}$ mapped on to them under $\mathcal{M}$. Clearly, S' must be of smaller length than S (S' will have m elements). It is also easy to see that $\mathcal{M}$ is a mapping function from $\mathcal{P}$ to S'. Finally, S' must satisfy all all the monotonic auxiliary constraints of $\mathcal{P}$ under $\mathcal{M}$. (To see this, suppose there is an monotonic auxiliary constraint $\mathcal{C}$ that S' does not satisfy under $\mathcal{M}$. By definition of monotonic auxiliary constraints (Section 3.4.2), this is impossible, since S, which is obtained by adding operators to S', satisfies all the monotonic auxiliary constraints under $\mathcal{M}$.[11]) This

[11] This is the primary reason for defining candidate sets only in terms of monotonic auxiliary constraints.

shows that S' must also be a candidate of $\mathcal{P}$ under the mapping function $\mathcal{M}$, which will violate the hypothesis that S is a minimal candidate. □

We will now continue with the proof of the correspondence theorem. Suppose S is a minimal candidate of $\mathcal{P}$ with respect to the mapping function $\mathcal{M}$. By the lemma above, S has as many elements as steps in $\mathcal{P}$. This makes $\mathcal{M}$ a one-to-one mapping from steps of the plan to elements of S (excepting t_0 and t_∞). Consider the step sequence G' obtained by translating the operators in S to step names under the mapping $\mathcal{M}^{-1}$ such that $G'[i] = \mathcal{M}^{-1}(S[i])$ (note that $\mathcal{M}$ can be inverted since it is a one-to-one mapping). Let G be the step sequence obtained by adding t_0 to the beginning and t_∞ to the end of G'. Since $\mathcal{M}$ maps all steps of $\mathcal{P}$ (except t_0 and t_∞) to elements of S, G' will contain all steps of $\mathcal{P}$. Further, since by the definition of mapping function, S satisfies all the ordering relations of $\mathcal{P}$ under $\mathcal{M}$, G also satisfies all the ordering relations of $\mathcal{P}$. This makes G a ground linearization of $\mathcal{P}$ that corresponds to S. Since S also satisfies the auxiliary monotonic constraints of $\mathcal{P}$ under $\mathcal{M}$, by construction G must satisfy them too. Thus, G is a safe ground linearization that corresponds to the minimal candidate S. □

Example: In Section 3.4, we noted that $S_1 : o_1 o_2$ is a minimal candidate for the example plan $\mathcal{P}_E$. Earlier in this section, we also noted that safe ground linearization $G_1 : t_0 t_1 t_2 t_\infty$, and that G_1 corresponds to S_1.

We now have a strong connection between the syntactic concept, safe ground linearization, and the semantic concept, minimal candidate. This gives us a way of interpreting the meaning of the syntactic operations performed on the partial plans in terms of the candidate set of the partial plan. Checking whether a partial plan has an empty candidate set can be done by checking if it has a safe ground linearization:

Proposition 20 *A partial plan has an empty candidate set (and is inconsistent) if and only if it has no safe ground linearizations.*

This follows directly from the correspondence theorem. Similarly, checking whether a minimal candidate of the partial plan is a solution to the problem can be done by looking at the ground operator sequences corresponding to the safe ground linearizations.

4 A generalized algorithm for Partial Order Planning

The algorithm `Refine-Plan` in Figure 7 instantiates the refinement search (Figure 2) within the context of planning. In particular, it describes a generic refinement-planning algorithm, the specific instantiations of which cover most of

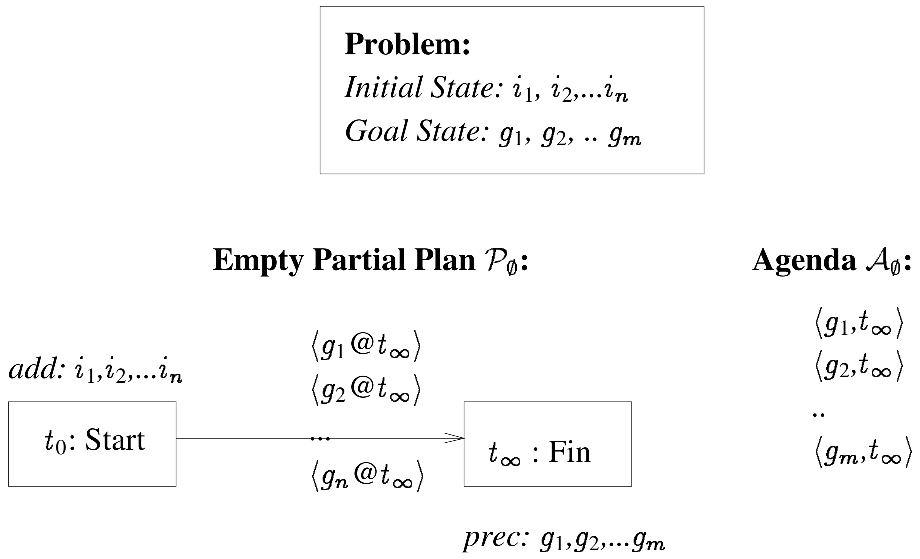


Fig. 6. The ''empty'' partial plan $\mathcal{P}_\emptyset$ and the agenda with which `Refine-plan` is first invoked.

the partial-order plan-space planners. [12]

As we noted in Section 3.4.4, the main refinement operation of `Refine-Plan`, called establishment refinement, is to consider each PTC (corresponding to some precondition of some step of the plan) in turn and work towards adding constraints to the partial plan so that all of its minimal candidates will satisfy that PTC. Accordingly, each invocation of `Refine-Plan` takes a partial plan, along with a datastructure called *agenda* that keeps track of the set of PTCs still to be considered for establishment during planning. Given a planning problem [I,G], where G is a set of goals (of attainment), the planning process is initiated by invoking `Refine-Plan` with the ''null'' partial plan $\mathcal{P}_\emptyset$ and the agenda $\mathcal{A}_\emptyset$ where

$$\mathcal{P}_\emptyset : \langle \{t_0, t_\infty\}, \{t_0 \prec t_\infty\}, \emptyset, \{t_0 \rightarrow \texttt{start}, t_\infty \rightarrow \texttt{fin}\}, \mathcal{L}_\emptyset : \{\langle g_i@t_\infty\rangle | g_i \in G\}\rangle$$

and

$$\mathcal{A}_\emptyset : \{\langle g_i, t_\infty\rangle | g_i \in G\}$$

where corresponding to each goal $g_i \in G$, $\mathcal{A}_\emptyset$ contains $\langle g_i, t_\infty\rangle$, and $\mathcal{L}_\emptyset$ contains the PTC $\langle g_i@t_\infty\rangle$. Figure 6 illustrates $\mathcal{P}_\emptyset$ and $\mathcal{A}_\emptyset$. [13] As noted earlier, the candidate

[12] An important exception is the hierarchical task reduction planners, such as SIPE [40], IPEM [1] and O-Plan [5]. However, see [17] for a discussion of how `Refine-Plan` can be extended to cover these planners.

[13] Alert readers may note that there is some overlap between the agenda, and the definition of PTCs. Agenda is a prescriptive datastructure used by the planner to keep track of the

set of $\mathcal{P}_\emptyset$, $\langle\langle\mathcal{P}_\emptyset\rangle\rangle$ is the candidate space of the problem $\mathcal{K}$.

The procedure `Refine-Plan` specifies the refinement operations done by the planning algorithm. Comparing this algorithm to the refinement search algorithm in Figure 2, we note that it uses two broad types of refinements: the establishment refinements mentioned earlier (step 1); and the tractability refinements (step 3) to be discussed in Section 4.5. In each refinement strategy, the added constraints include step addition, ordering addition, binding addition, as well as addition of auxiliary constraints. In the following sub-sections, we briefly review the individual steps of this algorithm.

Table 1 characterizes many of the well known plan-space planners as instantiations of the `Refine-Plan` algorithm. `Refine-Plan` is *modular* in that its individual steps can be analyzed and instantiated relatively independently. Furthermore, the algorithms do not assume any specific restrictions on action representation, and can be used by any planner using the ADL action representation [29]. Although we will be concentrating on goals of attainment, other richer types of behavioral constraints, such as maintenance goals, and intermediate goals, can be handled by invoking `Refine-Plan` with a plan that contains more initial constraints than $\mathcal{P}_\emptyset$ described above (see [15]). In particular, maintenance goals can be handled by imposing some interval preservation constraints on the initial plan. Similarly intermediate goals can be handled by introducing some dummy steps (in addition to t_0 and t_∞) into the plan, and introducing the intermediate goals as PTCs with respect to those steps.

4.1 Solution Constructor function

As discussed in Section 3, the job of a solution-constructor function is to look for and return a solution from the candidate set of a partial plan. [14] Since enumerating and checking the full candidate set of a partial plan is infeasible, most planners

---

preconditions that need to be established. The agenda does not affect the candidate set of the partial plan. The PTCs, in contrast, are only checked to see if a candidate is a solution. Under this model, the planner can terminate without having explicitly considered each of the preconditions in the agenda (as long as all the auxiliary constraints, including the PTCs are satisfied). Similarly, it also allows us to post preconditions that we do not want the planner to explicitly work on. In particular, the so called ''filter-conditions'' [4,39] can be modeled by adding them to the PTCs, without adding them to the agenda. This is in contrast to ordinary preconditions which are added to both the agenda, and the auxiliary constraints.

[14] 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.

Algorithm Refine-Plan($\langle \mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle, \mathcal{A} \rangle$) **Parameters**: ($i$) `sol`: Solution constructor function. (The following parameters are used by the refinement strategies) (ii) `pick-prec`: the routine for picking the preconditions from the plan agenda for establishment. (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 monotonic auxiliary constraints.

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

1. Establishment Refinement: Refine the plan by selecting a goal, choosing a way of establishing that goal, and optionally remembering the establishment decision.

1.1 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 work on. *Not a backtrack point.*

1.2. 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.*

1.3. Bookkeeping: (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.

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

2.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.*

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

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

4. Recursive Invocation: Recursively invoke `Refine-plan` on the refined plan.

Fig. 7. A generalized refinement algorithm for plan-space planning

concentrate instead on the minimal candidates. As discussed in Section 3, (see Proposition 19 and Figure 3), it is possible to get a complete refinement search in the space of ground operator sequences if we have a solution constructor which examines the safe ground linearizations and see if any of those correspond to a solution. This leads to the prototypical solution constructor, *Some-sol*:

Definition 21 (Some-sol) *Given a partial plan $\mathcal{P}$, return with success when some ground linearization G of $\mathcal{P}$ also satisfies all the PTC's (this means that the ground operator sequence S corresponding to G isa solution to the problem)*

It is possible to show that any instantiation of `Refine-Plan` using *Some-sol* leads to a complete refinement search, *as long as the refinement operators used by the planner are complete* (Defn. 1). Unfortunately, implementations of *Some-sol* are not in general tractable. [15] Because of this, most implemented planners use a significantly restricted version of *Some-sol* called *All-sol*, which terminates only when *all* ground linearizations are safe and *all* safe ground linearizations correspond to solutions (satisfy all PTCs):

Definition 22 (All-sol) *Given a partial plan $\mathcal{P}$, return with success only when all ground linearizations of the plan are safe, and all safe ground linearizations satisfy all the PTCs. (This means that the ground operator sequences corresponding to all safe ground linearizations are solutions).*

The solution constructors used by most existing planners correspond to some implementation of *All-sol*, and the completeness proofs of these planners are given in terms of *All-sol*.

Comparing *All-sol* to the definition of a solution constructor (Defn. 2), we note that in general, *All-sol* requires the partial plan to have more than one solution before it will signal success (all minimal candidates must be solutions). One theoretically inelegant consequence of this difference is that for planners using *All-sol* as the solution constructor, completeness of refinement operators alone does not guarantee the completeness of refinement search. In Section 5.2, we describe some specific instantiations of `Refine-plan` that illustrate this.

In particular, an instantiation of `Refine-Plan` that uses complete refinement operators, and uses *All-sol* based solution constructor will be complete *only if the refinements eventually produce a partial plan all ground linearizations of which correspond to solutions* (i.e., safe, and satisfy all PTCs). [16] We will note later that this is in general ensured as long as the planner either uses tractability refinements (Section 4.5), or continues to use establishment refinements as long

[15] this is related to the fact that possible correctness of a partially ordered plan is NP-hard [3,14]

[16] Note that this needs to happen for at least one partial plan, not necessarily all partial plans.

as there are conditions that not yet necessarily true (see description of TWEAK in Section 5.1.3).

Once a planner is complete for *All-sol*, it is actually possible to use slightly more general versions of *All-sol*, called *k-sol*, which randomly check k safe ground linearizations of the plan to see if any of them are solutions. If a planner is complete for *All-sol*, it is also complete for *k-sol*. This is because completeness with respect to *All-sol* means that eventually a partial plan is produced all of whose ground linearizations become safe, and will correspond to solutions. When this happens, *k-sol* will also terminate with success on that partial plan. Further, *k-sol* is guaranteed to terminate the search before *All-sol*.

The termination criteria of *All-sol* correspond closely to the notion of *necessary correctness* of a partially ordered plan, first introduced by Chapman [3]. Existing planning systems implement *All-sol* in two different ways: Planners such as Chapman's TWEAK [3,42] use the modal truth criterion to explicitly *check* that all the safe ground linearizations correspond to solutions (we will call these the MTC-based constructors). Planners such as SNLP [24] and UCPOP [33] depend on protection strategies and conflict resolution (Section 4.5.2) to indirectly guarantee the safety and necessary correctness required by *All-sol* (we call these protection based constructors). In this way, the planner will never have to explicitly reason with all the safe-ground linearizations.

4.2 Goal Selection and Establishment

As we noted in Section 3.4.4 the fundamental refinement operation used in refinement planners is the so-called establishment operation which adds constraints to the plan so that its minimal candidates satisfy all the PTCs. The establishment refinement involves selecting a precondition $\langle c, s\rangle$ of the plan from the agenda (where c is a precondition of a step s), and refining (i.e., adding constraints to) the partial plan such that in each refinement some step t gives c, and c is not violated by any steps intervening between t and s. When this is done, it is easy to see that all the minimal candidates of the resulting plan will satisfy the PTC $\langle c@s\rangle$ (Section 3.4.2). Different refinements correspond to different steps acting as contributors of c to s. Chapman [3] and Pednault [29] provide theories of sound and complete establishment refinement. Pednault's theory is more general as it deals with actions containing conditional and quantified effects. [17] It is possible to limit `Refine-Plan` to establishment refinements alone and still get a sound and complete (in the sense of Definition 3) planner (using either *Some-sol* or *all-sol* described earlier as solution constructors).

[17] And it also separates checking the truth of a proposition from planning to make that proposition true, see [14].

Planner	Soln. Constructor	Goal Selection	Bookkeeping	Tractability Refinements
Existing Planners				
TWEAK [3]	MTC-based $O(n^4)$	MTC-based $O(n^4)$	None	None
UA [27]	MTC-based $O(n^2)$	MTC-based $O(n^2)$	None	Unambiguous ordering
Nonlin [39]	MTC (Q&A) based	Arbitrary $O(1)$	Interval & Goal Protection	Conflict Resolution
TOCL [2]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Total ordering
Pedestal [23]	Protection based $O(1)$	Arbitrary $O(1)$	Interval Protection	Total ordering
SNLP [24] UCPOP [33]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Conflict resolution
MP, MP-I [12]	Protection based	Arbitrary	(Multi) contributor protection	Conflict resolution
Hybrid Planners (described in Section 5.2)				
SNLP-UA	MTC based $O(n^2)$	MTC based $O(n^2)$	Contributor protection	Unambiguous Ordering
SNLP-MTC	MTC based $O(n^4)$	MTC based $O(n^4)$	Contributor protection	conflict resolution
SNLP-CON	MTC based $O(n^4)$	MTC based $O(n^4)$	Contributor protection	*None*
McNONLIN-MTC	MTC based $O(n^4)$	MTC based $O(n^4)$	Interval protection	conflict resolution
McNONLIN-CON	MTC based $O(n^4)$	MTC based $O(n^4)$	Interval protection	*None*
TWEAK-visit	MTC-based $O(n^4)$	MTC-based $O(n^4)$	Agenda popping	None

Table 1
Characterization of a variety of existing as well as hybrid planners as instantiations of `Refine-Plan`. (The n used in the complexity figures is the number of steps in the partial plan)

In Pednault's theory, establishment of a condition c at a step s essentially involves selecting some step s' (either existing or new), and adding enough constraints to the plan such that (i) $s' \prec s$, (ii) s' causes c to be true and (iii) c is not violated before s. To ensure ii, we need to, in general, ensure the truth of certain additional conditions before s'. Pednault calls these the *causation preconditions* of s' with respect to c. To ensure iii, for every step s'' of the plan, we need to either make s'' come before s', or make s'' come after s, or make s'' necessarily preserve c. The last involves guaranteeing truth of certain conditions before s''. Pednault calls these the *preservation preconditions* of s'' with respect to c. Causation and preservation preconditions are called the *secondary preconditions* of the action. These become PTCs of the partial plan (for each secondary precondition c of s, add the PTC $\langle c@s \rangle$), and are also added to the agenda data structure (to be considered by establishment refinement later).

Goal Selection: The strategy used to select the particular precondition $\langle c, s \rangle$ to be established, (called the goal selection strategy) can be arbitrary, can depend on some ranking based on precondition abstraction [18,34], and/or demand driven (e.g. select a goal only when it is not already necessarily true according to the modal truth criterion [3]). The last strategy, called MTC-based goal selection, involves reasoning about the truth of a condition in a partially-ordered plan, and can be intractable for general partial orderings consisting of ADL [29] actions (see Table 1, as well as the discussion of pre-ordering strategies in Section 4.5.1.).

4.3 Bookkeeping and Protecting establishments

It is possible to do establishment refinement without bookkeeping step. Chapman's TWEAK [3] is such a planner. However, such a planner is not guaranteed to respect its previous establishment decisions while making new ones, and thus may have a high degree of redundancy. Specifically such a planner may:

(i) wind up visiting the same candidate (potential solution) in more than one search branch (in terms of our search space characterization, this means $\rho > 1$), and
(ii) wind up having to consider the same precondition (PTC) for establishment more than once.

The examples in Figure 8 illustrate both these behaviors on a planner that only uses establishment refinement. Example 8(a) (originally from Minton et al. [26]) shows that a planner without any form of book-keeping may find the same solution in multiple different search branches. (That is, the candidate sets of the search nodes in different branches overlap.) Specifically, the ground operator sequence $o_3 o_2 o_1$ belongs to the candidate sets of nodes both in the left and right branches

.

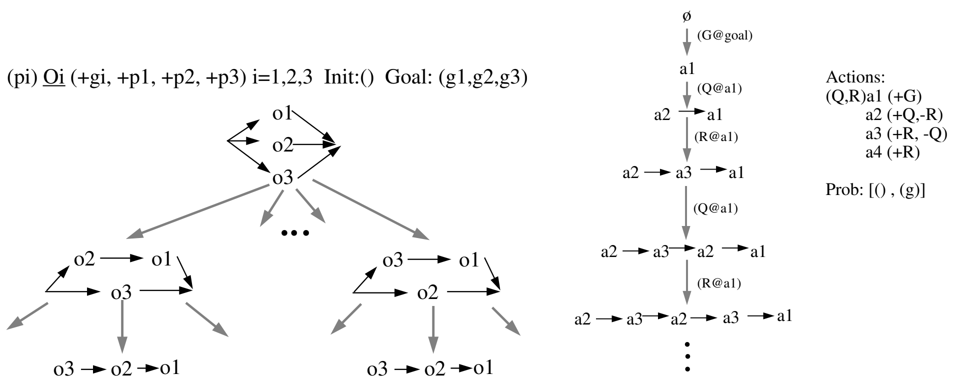
Protection Method	Implementation	Refine-Plan Property
Agenda popping	When the precondition $\langle p, s\rangle$ is considered for establishment, remove it from the agenda. Prune any partial plan whose agenda is empty.	Will not consider the same precondition for establishment more than once.
Interval Protection	Add an IPC $\langle s', p, s\rangle$ to the auxiliary constraints of the plan, whenever a precondition $\langle p, s\rangle$ is established through the effects of s'.	Same as above, but facilitates earlier pruning of plans that will ultimately necessitate the re-establishment of a condition that has already been considered for establishment.
Contributor Protection	Add two IPCs $\langle s', p, s\rangle$ and $\langle s', \neg p, s\rangle$ to the auxiliary constraints of the plan, whenever a precondition $\langle p, s\rangle$ is established through the effects of s'.	In addition to the properties of agenda popping and interval protection, it also ensures that the establishment refinement is *systematic* [24] (see Definition 1).
Multi-contributor Protection	Add the disjunctive IPC $\langle s', p, s\rangle \vee \langle s'', p, s\rangle$ to the auxiliary constraints of the plan, whenever a commitment is made to establish the precondition $\langle p, s\rangle$ with the effects of either s' or s''.	Avoids the excessive commitment to contributors inherent in the interval protection and contributor protection strategies. But sacrifices systematicity [12].

Table 2
Implementation and properties of some common protection (bookkeeping) strategies in terms of `Refine-Plan` framework.

of the search tree. In the example 8(b) after having established a PTC $\langle Q@a_1\rangle$, the planner works on the PTC $\langle R@a_1\rangle$. In the resulting plan, the first PTC is no longer satisfied by any of the minimal candidates (this is typically referred to as ‘‘clobbering’’ of the precondition $\langle Q, a_1\rangle$). This means that $\langle Q@a_1\rangle$ needs to be established again. The bookkeeping step attempts to reduce these types of redundancy. Table 2 summarizes the various bookkeeping strategies uses by the existing planners.

At its simplest, the bookkeeping may be nothing more than removing each precondition from the agenda of the partial plan once it is considered for establishment. Since the establishment refinement looks at all possible ways of establishing a condition at the time it is considered, when the agenda of a partial plan is empty, it can be pruned without loss of completeness. We will call this strategy the *agenda popping* strategy. The hybrid planner TWEAK-Visit in Table 1, a variant of TWEAK, uses this strategy.

(a) (b)



(a) Considering the same candidate in more than one branch

(b) Establishing the same condition more than once

Fig. 8. Examples showing redundancy and looping in TWEAK search space. In all the examples, the operators are shown with preconditions on the right side, and effects on the left (with ''+'' sign for add list literals, and ''−'' sign for delete list literals). The Init and Goal lists specify the problem. The example on left is adopted from Minton et al.

A more active form of bookkeeping involves protecting previous establishments in a partial plan, while making new refinements to it. In terms of `Refine-Plan`, such protection strategies can be seen as posting IPCs on the partial plan to record the establishment decisions. The intuition behind this is that the IPCs will constrain the candidate set of the plan such that ground operator sequences corresponding to partial plan ground linearizations that do not satisfy the PTC are automatically removed from the candidate set (by making the corresponding ground linearizations unsafe). When there are no safe ground linearizations left, the plan can be abandoned without loss of completeness (even if its agenda is not empty).

The protection strategies used by classical partial order planners come in two main varieties: interval protection[18], and contributor protection[19] 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 [23]), the bookkeeping 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 the interval preservation constraint $\langle s', p, s \rangle$. Finally, for bookkeeping 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.[20] This contributor protection can be modeled in terms of the twin interval

[18] also called causal link protection, or protection intervals in the literature

[19] also called exhaustive causal link protection [12]

[20] See [11] for a reconstruction of the ideas underlying goal protection strategies.

preservation constraints $\langle s', p, s\rangle$ and $\langle s', \neg p, s\rangle$. [21]

While most planners use one or the other type of protection strategies exclusively for all conditions, planners like NONLIN and O-Plan [39,5] post different bookkeeping constraints for different types of conditions. Finally, the interval protections and contributor protections can also be generalized to allow for multiple contributors supporting a given condition [12]. In particular, a multiple-contributor protection may represent the commitment that the preconditi on p of step s' will be given by either s_1 or s_2. Such a protection can be represented as a disjunction of two IPCs: $\langle s_1, p, s'\rangle \vee \langle s_2, p, s'\rangle$.

4.3.1 Contributor Protections and Systematicity

While all the bookkeeping strategies described above avoid considering the same precondition for establishment more than once (and thus avoid the looping described in Figure 8(b)), only the contributor protection eliminates the redundancy of overlapping candidate sets, by making establishment refinement systematic. Specifically, we have:

Proposition 23 (Systematicity of establishment with Contributor Protection[24]) *Establishment refinement with contributor protection is systematic in that partial plans in different branches of the search tree will have non-overlapping candidate sets (thus $\rho = 1$).*

This property can be proven from the fact that contributor protections provide a way of uniquely naming steps independent of the symbol table mapping (see [24,11]). To understand this, consider the following partial plan (where the PTCs are omitted for simplicity):

$$\mathcal{N} : \left\langle \begin{array}{l} \{t_0, t_1, t_\infty\}, \{t_0 \prec t_1, t_1 \prec t_\infty\}, \emptyset, \\ \{t_1 \to o_1, t_0 \to \mathtt{start}, t_\infty \to \mathtt{fin}\}, \\ \{\langle t_1, p, t_\infty\rangle, \langle t_1, \neg p, t_\infty\rangle\} \end{array} \right\rangle$$

where the step t_1 is giving condition p to t_∞, the goal step. Suppose t_1 has a precondition q. Suppose further that there are two operators o_2 and o_3 respectively in the domain which can both provide the condition q. The establishment refinement

[21] It is easy to see that contributor protection implies interval protection. What is not obvious at first glance is that in the presence of the optional consistency check, it also implies the essence of agenda popping (in that it will not allow the planner to consider the same precondition for establishment more than once).

generates two partial plans:

$$\mathcal{N}_1 : \left\langle \begin{array}{l} \{t_0, t_1, t_2, t_\infty\}, \{t_0 \prec t_2, t_2 \prec t_1, t_1 \prec t_\infty\}, \emptyset, \\ \{t_1 \to o_1, t_2 \to o_2, t_0 \to \texttt{start}, t_\infty \to \texttt{fin}\}, \\ \{\langle t_1, p, t_\infty\rangle, \langle t_1, \neg p, t_\infty\rangle, \langle t_2, q, t_\infty\rangle, \langle t_2, \neg q, t_\infty\rangle\} \end{array} \right\rangle$$

$$\mathcal{N}_2 : \left\langle \begin{array}{l} \{t_0, t_1, t_2, t_\infty\}, \{t_0 \prec t_2, t_2 \prec t_1, t_1 \prec t_\infty\}, \emptyset, \\ \{t_1 \to o_1, t_3 \to o_3, t_0 \to \texttt{start}, t_\infty \to \texttt{fin}\}, \\ \{\langle t_1, p, t_\infty\rangle, \langle t_1, \neg p, t_\infty\rangle, \langle t_2, q, t_\infty\rangle, \langle t_2, \neg q, t_\infty\rangle\} \end{array} \right\rangle$$

Consider the step t_2 in $\mathcal{N}_1$. This can be identified independent of its name in the following way:

> ''The step which gives q to the step which in turn gives p to the dummy final step''

An equivalent identification in terms of candidates is:

> ''The last operator with an effect q to occur before the last operator with an effect p in the candidate (ground operator sequence)''

The contributor protections ensure that this operator is o_2 in all the candidates of $\mathcal{N}_1$ and o_3 in all the candidates of $\mathcal{N}_2$. Because of this, no candidate of $\mathcal{N}_1$ can ever be a candidate of $\mathcal{N}_2$, thus ensuring systematicity of establishment refinement.

The discussion about bookkeeping strategies in this section demonstrates that systematicity should not be seen in isolation, but rather as part of a spectrum of methods for reducing redundancy in the search space. We will return to this motif in Section 7.2.

4.4 Consistency Check

The aim of the consistency check is to prune inconsistent partial plans (i.e., plans with empty candidate sets) from the search space, thereby improving the performance of the overall refinement search. (Thus, from the completeness point of view, consistency check is an optional step.) Given the relation between the safe ground linearizations and candidate sets (Proposition 20), the consistency check can be done by ensuring that each partial plan has at least one safe ground linearization. This requires checking the consistency of orderings, bindings and auxiliary constraints of the plan. Ordering consistency can be checked in polynomial time, binding consistency is tractable for infinite domain variables, but is intractable for finite domain variables. Finally, consistency with respect to auxiliary constraints is also intractable for many common types of monotonic auxiliary constraints (even for ground partial plans without any variables). Specifically, we have:

Proposition 24 *Given a partial plan whose monotonic auxiliary constraints contain interval preservation constraints, checking if there exists a safe ground linearization of the plan is NP-hard.*

This proposition directly follows from a result due to Smith [35], which shows that checking whether there exists a conflict-free ground linearization of a partial plan with interval preservation constraints is NP-hard.

4.5 Tractability refinements

Since, as observed above, the consistency check is NP-hard in general, each call to `Refine-Plan` is also NP-hard. It is of course possible to reduce the cost of refinement by pushing the complexity into search space size. In general, when checking the satisfiability of a set of constraints is intractable, we can still achieve polynomial consistency check by refining the partial plans into a set of mutually exclusive and exhaustive constraint sets such that the consistency of each of those refinements can be checked in polynomial time

This is the primary motivation behind tractability refinements. To understand the type of refinements that need to be done, we note that the reason for the intractability of consistency check is that checking whether a plan has a safe ground linearization (Proposition 20) requires going through a potentially exponential number of ground linearizations. Thus, to make it tractable, we could either restrict each partial plan to have less than exponential ground linearizations, or make all the ground linearizations be uniform with respect to their satisfaction of the IPCs (Defn. 11) -- e.g., either all of them satisfy an IPC or none of them satisfy it. In the later case, the consistency can be checked by looking at any one ground linearization. These ideas are realized in two broad classes of tractability refinements: *pre-ordering* and *conflict resolution*.

Note that an interesting property of partial plans in the presence of tractability refinements is that eventually the refinements will produce a partial plan all of whose ground linearizations are safe. This later property, coupled with the fact that all planners will eventually consider each PTC for establishment once, ensures that any instantiation of `Refine-Plan` which uses tractability refinements will eventually produce a partial plan all of whose ground linearizations of a partial plan correspond to solutions.[22] Thus, they will be complete with respect to *All-sol* based solution constructors (see Section 4.1).

[22] This claim assumes the completeness of establishment refinements

4.5.1 Pre-ordering refinements

Pre-ordering strategies aim to restrict the type of partial orderings in the plan such that consistency with respect to monotonic auxiliary constraints can be checked without explicitly enumerating all the ground linearizations. Two possible pre-ordering techniques are *total ordering* and *unambiguous ordering* [26]. Total ordering orders every pair of steps in the plan, while unambiguous ordering orders a pair of steps only when one of the steps has an effect c, and the other step either negates c or needs c as a precondition (implying that the two steps *may* interact). Both of them guarantee that in the refinements produced by them, either all ground linearizations will be safe or none will be. [23] Thus, consistency can be checked in polynomial time by examining any one ground linearization.

Pre-ordering techniques can also make other plan handling steps, such as MTC-based goal selection and MTC-based solution constructor, tractable (c.f. [26,11]). For example, unambiguous plans also allow polynomial check for necessary truth of any condition in the plan. Polynomial necessary truth check can be useful in MTC-based goal selection and termination tests. In fact, unambiguous plans were originally used in UA [26] for this purpose.

4.5.2 Conflict Resolution Refinements

Conflict resolution refines a given partial plan with the aim of compiling the monotonic auxiliary constraints into the ordering and binding constraints. Specifically, the partial plan is refined (by adding ordering, binding or secondary preconditions [29] to the plan) until each possible violation of the auxiliary candidate constraint (called conflict) is individually resolved.

An interval preservation constraint $\langle s_i, p, s_j \rangle$ is threatened by a step s' can possibly come between s_i and s_j and not preserve p (note from Defn. 11 that this means that at least one ground linearization of the plan will not satisfy the IPC). A conflict is specified by an IPC and a threatening step.

Resolving the conflict involves either making s' not intervene between s_i and s_j (by adding either the ordering $s' \prec s_i$ or the ordering $s_j \prec s'$), or adding secondary (preservation) preconditions of s', required to make s' preserve c [29], as PTCs to the partial plan (and to the agenda). The ordering strategies for resolving conflicts are called *promotion* and *demotion*, while the secondary precondition based conflict resolution is called *confrontation*. When all conflicts are resolved this way, the resulting refinements will have the property that all their ground linearizations are safe (or will eventually become safe in the case of confrontation). Thus, checking the partial plan consistency will amount to checking for the existence of ground

[23] In the case of total ordering, this holds vacuously true since the plan has only one linearization

linearizations. This can be done by checking ordering and binding consistency.

4.5.3 Deferring Tractability Refinements

Until now, we assumed that the only choice regarding tractability refinements is to either use them, or not use them. However, since the tractability refinements are optional, it is also possible to do *some* tractability refinements, while ignore or defer other refinements. For example, we could pre-order *some* potentially conflicting steps, while leaving others unordered. Similarly, we can resolve *some* conflicts, while leaving others unresolved. Finally, we could either handle the unordered and/or unresolved conflicts by the end of the search, or ignore them all together. Such strategies could be useful in improving performance [32], since as we shall see in Section 6, tractability refinements reduce the cost of consistency check at the expense of increased branching factor (corresponding to additional refinements). This type of selective use of tractability refinements does not in general affect soundness and completeness of the `Refine-Plan`. There are two caveats however:

(i) If the tractability refinements are being *ignored* rather than being *deferred*, then in general `Refine-Plan` is not guaranteed to produce a partial plan all of whose ground linearizations are safe. This means that for such instantiation of `Refine-Plan`, completeness with respect to solution constructors based on *All-sol* is not guaranteed (Section 4.1).
(ii) If the `Refine-Plan` uses a consistency check based only on orderings and bindings, then it may wind up not detecting the inconsistency of a partial plan (specifically, the deferred/ignored conflicts may be unresolvable). This means that `Refine-Plan` could refine inconsistent plans, thereby unnecessarily increasing the search space size in the worst case. In particular, `Refine-Plan` is not guaranteed to be *informed* and consequently will not be *strongly systematic* (Definition 7). (This may not necessarily have any impact on the performance of the planner however, see Section 7.2).

5 Specific Instantiations of Refine-Plan

As we observed in the previous section, most existing partial-order planning algorithms can be seen as instantiations of `Refine-Plan` algorithm. Table 1 characterizes many well-known algorithms in terms of the way they instantiate the different parts of the `Refine-Plan` algorithm. To make things concrete, and to help in focused empirical comparisons in the later sections, we will now provide more details about some specific instantiations of `Refine-Plan`. We will first discuss instantiations that correspond to some well-known existing partial-order planners and then (in Section 5.2) discuss some instantiations of `Refine-Plan` that have not been previously discussed in the literature.

5.1 Existing Planners

In this section we will discuss the instantiations of `Refine-Plan` corresponding to four well-known planners: SNLP [24], McNONLIN [39], TWEAK [3] and UA [27]. We will start with the instantiation of `Refine-Plan` that corresponds to SNLP, called `Refine-Plan-SNLP`, and describe the other three algorithms in terms of the incremental changes that need to be made to the SNLP instantiation.

5.1.1 The SNLP Algorithm

The following describes the SNLP algorithm, first described in [24], as an instantiation of `Refine-Plan`.

Algorithm `Refine-Plan-SNLP`$(\langle \mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle, \mathcal{A} \rangle)$

0. Termination: If $\mathcal{A}$ is empty, report success and stop.

1.1. Goal selection: Pick any $\langle p, s_{\text{need}} \rangle \in \mathcal{A}$. Set $\mathcal{A} = \mathcal{A} - \langle p, s_{\text{need}} \rangle$

1.2. Establishment:

-- Let s_{add} be an existing step, or some new step, that adds p before s_{need}. If no such step exists or can be added then backtrack. Set $T = T \cup \{s_{\text{add}}\}$, $O = O \cup \{s_{\text{add}} \prec s_{\text{need}}\}$, $and\ \mathcal{B} = \mathcal{B} \cup$ the set of variable bindings (causation preconditions) to make s_{add} add p.

-- For every step s_t such that s_t comes between s_{add} and s_{need} and deletes p, make two refinements, one in which $O = O + (s_{\text{need}} \prec s_t)$ and the other in which $O = O + (s_t \prec s_{\text{add}})$.

-- Finally, if s_{add} is new, then update the agenda and the set of non-monotonic auxiliary constraints:

$$\mathcal{A} = \mathcal{A} \cup \{\langle p', s_{\text{add}} \rangle | \forall p' \in precond(s_{\text{add}})\}$$

and

$$\mathcal{L} = \mathcal{L} \cup \{\langle p' @ s_{\text{add}} \rangle | \forall p' \in precond(s_{\text{add}})\}$$

(For completeness, all ways of establishing the condition must be considered.)

1.3. Bookkeeping: Update the set of monotonic auxiliary constraints with two interval preservation constraints:

$$\mathcal{L} = \mathcal{L} + \langle s_{\text{add}}, p, s_{\text{need}} \rangle + \langle s_{\text{add}}, \neg p, s_{\text{need}} \rangle$$

2.b. Conflict Resolution (Tractability Refinements):

-- *Threat/Conflict detection:* A step t is said to be a threat for an interval preservation constraint $\langle s, p, s' \rangle$ of the plan, if $t \in T$, $\langle s, p, s' \rangle \in \mathcal{L}$, such that

t can possibly come between s and s', and the effect of t necessarily violates (does not preserve) p.

-- *Threat Resolution:* For each threat consisting of step t and IPC $\langle s, p, s' \rangle$, make two refinements, one in which $\mathcal{O} = \mathcal{O} + (t \prec s)$ and the other in which $\mathcal{O} = \mathcal{O} + (s' \prec t)$.

3. Consistency Check: Prune the plan if it is either order-inconsistent ($\mathcal{O}$ contains cycles) or is binding inconsistent ($\mathcal{B}$ contains both a codesignation and a non-codesignation between a pair of variables).

4. Recursive Invocation: Recursively invoke `Refine-Plan-SNLP` on the refined plan

Note that the consistency check in the above algorithm checks order and binding consistency, rather than the existence of safe ground linearizations (Section 3.5). However, as we discussed in Section 4.5, as long as we do complete conflict resolution with respect to auxiliary constraints, every (remaining) ground linearization of the plan is guaranteed to be a safe ground linearization. Thus, consistency check can be done by checking for the existence of ground linearizations (which is equivalent to ensuring that ordering and binding constraints are consistent). [24]

Most common implementations of SNLP, including the one we used in our experiments (Section 8) avoid a separate consistency check in step 4 by compiling the order and binding consistency checks into the establishment and conflict resolution refinements (thus ensuring that none of the refinements produced in either step 2 or step 4 are order or binding inconsistent). Further, some implementations of SNLP, such as the popular one by Barrett and Weld [2] do conflict resolution one threat per iteration (invocation of `Refine-Plan`). While this can also be cast as an instantiation of `Refine-Plan` (which defers tractability refinements; Section 4.5.3) the implementation we used in our experiments is faithful to the description above. [25]

In writing the algorithm above, we made the implicit assumption that the domain actions do not have conditional or quantified effects. The primary reason for making this assumption is the desire to make the exposition simple, rather than any fundamental limitation of the `Refine-Plan` algorithm. If the actions have conditional effects, the establishment step and the conflict resolution step must consider adding causation and preservation preconditions during establishment and threat resolution.

[24] Notice that our description of conflict resolution avoids the +ve threat -ve threat terminology commonly used in describing SNLP. This is because each ''causal link'' used by SNLP naturally corresponds to two independent IPCs, and the notion of threat need only be defined with respect to an IPC.

[25] The SNLP implementation used by Peot and Smith in their conflict deferment experiments [32] corresponds closely to our description above.

5.1.2 The McNONLIN Algorithm

SNLP is a descendant of NONLIN [39] planning algorithm. There are several differences between NONLIN, which is a hierarchical task reduction planner, and SNLP, which a non-hierarchical partial order planner. One difference that is of interest with respect to `Refine-Plan` is that while SNLP uses contributor protection, NONLIN used interval protections. [26] To illustrate this difference, we describe an algorithm called McNONLIN that is inspired by the protection strategies of NONLIN.

1.3. Bookkeeping: Update auxiliary constraint with one interval preservation constraint: $\mathcal{L} = \mathcal{L} + \langle s_{\text{add}}, p, s_{\text{need}} \rangle$ to the auxiliary constraints.

Note that McNONLIN adds fewer number of auxiliary constraints per establishment than SNLP does. This means that a search branch becomes inconsistent for SNLP earlier than it does for McNONLIN. In depth-first regimes, this means SNLP backtracks earlier (and more often) than McNONLIN (the flip side being that McNONLIN will have more redundancy in its search space than SNLP). [27]

The differences in the bookkeeping step also has an implicit effect on the conflict resolution step, since conflict resolution is done with respect to each auxiliary constraint. Specifically, McNONLIN will generate fewer tractability (conflict-resolution) refinements than SNLP.

5.1.3 The TWEAK Algorithm

The primary difference between TWEAK and the two previous algorithms is that TWEAK does not use any form of bookkeeping constraints. Thus it neither uses the bookkeeping step, nor the conflict resolution step. Further, the standard formulation of the TWEAK algorithm [3] uses the modal truth criterion (MTC) for goal selection as well as termination. In particular, MTC is used to check if a given precondition $\langle C, s \rangle \in \mathcal{A}$ is necessarily true in the plan. The goal selection step prefers conditions that are not yet necessarily true, and the termination test succeeds as soon as all the preconditions are necessarily true (i.e., there are no conditions that need to be established). The following describes the steps of `Refine-Plan` where TWEAK

[26] Actually, the original NONLIN used different types of protections based on the *type* of the precondition being protected [39]

[27] Note, once again, that by using IPCs to model the book-keeping constraints, we avoid having to redefine the notion of a threat.

differs from SNLP.[28]

0. Termination: If every precondition $\langle c, s\rangle \in \mathcal{A}$ is necessarily true according to the modal truth criterion, report success and stop.

1.1. Goal selection: Pick any $\langle p, s_{\text{need}}\rangle \in \mathcal{A}$ that is not necessarily true according to the modal truth criterion. *Do not remove* $\langle p, s_{need}\rangle$ *from* $\mathcal{A}$.

1.3. Bookkeeping: *None*

2. Tractability refinement: *None*

Notice that TWEAK never removes the condition from agenda when it is considered for establishment. This is what allows it to work on the same precondition more than once. Further, although TWEAK does not use any tractability refinements, it is still complete for *All-sol* (of which the MTC-based termination is a special case), since it continues to use the establishment refinement as long as there are preconditions of the plan that are not necessarily correct (according to MTC).

5.1.4 The UA Algorithm

Another partial-order planning algorithm that received a significant amount of analysis is Minton et al.'s UA planner [27]. UA is very similar to TWEAK (in that it uses no bookkeeping constraints) and employs goal-selection and termination strategies similar to TWEAK. The only difference between UA and TWEAK is that UA uses a pre-ordering tractability refinement. In particular, we can get UA algorithm by replacing Step 4 of TWEAK by the following:

2.a. Pre-ordering (Tractability Refinements):

-- **Interaction Detection:** A pair of steps s_1 and s_2 are said to be interacting with each other if s_1 and s_2 are not ordered with respect to each other (i.e., neither $s_1 \prec s_2$ nor $s_2 \prec s_1$) and
 · s_1 has a precondition p, and s_2 either has an effect p or an effect $\neg p$ **or**
 · s_2 has a precondition p, and s_1 either has an effect p or an effect $\neg p$ **or**
 · s_1 has an effect p and s_2 has an effect $\neg p$.
 Find every step s' in the plan such that s' interacts with s_{add}.

-- **Interaction Resolution:** For every step s' that interacts with s_{add}, either add the order $s' \prec s_{\text{add}}$ or the ordering $s_{\text{add}} \prec s'$. (Both orderings need to be considered for completeness).

[28] It is instructive to note that our formulation of TWEAK is not completely in correspondence with Chapman's [3] initial characterization of the algorithm. In particular, Chapman suggests that planning be done by inverting the modal truth criterion. Among other things, this involves using the so-called ''white-knight declobbering' clause during establishment. However, by Chapman's own admission, introducing new steps into the plan as white-knights greatly increases the branching factor and can thus be very inefficient. Accordingly, in our formulation, we do not allow new (white-knight) steps to be introduced during declobbering

The pre-ordering refinement used by UA entails an interesting property for the partial plans maintained by UA. All the partial plans produced at the end of UA's pre-ordering step are all *unambiguous* in the sense that every precondition $\langle c, s \rangle$ of the plan (where c is a condition that needs to be true at step s) is either necessarily true (i.e., true in all ground linearizations) or necessarily false (i.e., false in all ground linearizations). Because of this property, although UA uses MTC-based goal-selection and termination, the cost of interpreting MTC is smaller for the unambiguous partial plans maintained by UA ($O(n^2)$, where n is the number of steps in the plan) than for the plans maintained by TWEAK ($O(n^4)$) (see Table 1). In particular, necessary truth of a condition can be determined by simply examining any one ground linearization of the plan.

5.2 Hybrid Algorithms

In looking at the four existing partial-order planning algorithms as instantiations of `Refine-Plan`, we note that there are many opportunities for integrating the algorithms to make hybrid planning algorithms. In this section, we discuss four such instantiations of `Refine-Plan`, SNLP-MTC, McNONLIN-MTC, SNLP-UA and TWEAK-visit.

5.2.1 The SNLP-MTC, McNONLIN-MTC Algorithms

SNLP-MTC algorithm is similar to SNLP algorithm, except that it uses the goal selection and termination steps of TWEAK. Unlike TWEAK, which does not remove the condition from the agenda once it is considered for establishment, SNLP-MTC, like SNLP, does remove the condition from the agenda. In the same vein, McNONLIN-MTC is similar to McNONLIN, except that it uses the goal selection and termination steps of TWEAK.

5.2.2 The SNLP-UA Algorithm

SNLP-UA algorithm is similar to SNLP-MTC algorithm except that it uses a variation on the pre-ordering step of UA, instead of conflict resolution step. It thus borrows the bookkeeping step from SNLP, and the tractability refinement step from UA. In particular, the following algorithm shows SNLP-UA as an instantiation of `Refine-Plan`.

Algorithm `Refine-Plan`$(\langle \mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle, \mathcal{A} \rangle)$

0. Termination: *Same as TWEAK*
1.1 Goal selection: *Same as TWEAK*
1.2. Establishment: *Same as SNLP*

1.3. Bookkeeping: *Same as SNLP*

2.a. Pre-ordering (Tractability Refinements):

-- **Interaction Detection:** (The following is the same as the step used by UA, except for the underlined part). A pair of steps s_1 and s_2 are said to be interacting with each other if s_1 and s_2 are not ordered with respect to each other (i.e., neither $s_1 \prec s_2$ nor $s_2 \prec s_1$) and

· s_1 has a precondition p, and s_2 either has an effect p or an effect $\neg p$ **or**

· s_2 has a precondition p, and s_1 either has an effect p or an effect $\neg p$ **or**

· s_1 has an effect p and s_2 has an effect $\neg p$ *or p*. (Note that the underlined clause is not present in the interaction detection step of UA).

Find every step s' in the plan such that s' interacts with s_{add}.

-- **Interaction Resolution:** For every step s' that interacts with s_{add}, either add the order $s' \prec s_{\text{add}}$ or the ordering $s_{\text{add}} \prec s'$. (Both orderings need to be considered for completeness).

3. Consistency Check: Prune the plan if it is either order-inconsistent ($\mathcal{O}$ contains cycles) or is binding inconsistent ($\mathcal{B}$ contains both a codesignation and a non-codesignation between a pair of variables), or it contains any auxiliary constraint with a conflict (i.e., an IPC $\langle s', p, s\rangle$ and a step s'' which falls in between s' and s and deletes (does not preserve) p).

The consistency check prunes the plan if any auxiliary constraint has a conflict. This is reasonable since after the pre-ordering step, any remaining conflicts are unresolvable (by promotion or demotion).[29] It also shows that not every planner which uses protection strategies is required to use conflict-resolution step.

Finally, note that unlike UA, the interaction detection step of SNLP-UA considers two steps to be interacting even if they share the same effect. This is required to ensure that all the auxiliary constraints are either necessarily safe or necessarily unsafe.

5.2.3 The TWEAK-visit Algorithm

TWEAK-visit is the same as TWEAK except that it does not work on a precondition that has already been considered for establishment. TWEAK-visit thus uses the ''agenda popping'' strategy that we described in Section 4.3. The only difference between TWEAK-visit and TWEAK algorithms is the goal-selection step:

1. Goal selection: Pick any $\langle p, s_{\text{need}}\rangle \in \mathcal{A}$ that is not necessarily true according to the modal truth criterion. Set $\mathcal{A} = \mathcal{A} - \langle p, s_{\text{need}}\rangle$.

[29] When we consider actions with conditional effects, the consistency check must be changed to prune the plan only when the conflict cannot be *confronted* [33], i.e., resolved by posting preservation preconditions as additional preconditions of the plan.

Specifically, unlike TWEAK which does not remove a condition from $\mathcal{A}$ once it is considered for establishment, TWEAK-visit does remove it. It is instructive to note that TWEAK-visit avoids the looping described in Figure 8(b).

Since TWEAK-Visit does not use tractability refinements, and unlike TWEAK, establishes each precondition at most once, depending on the order in which goals are considered for establishment, it could be incomplete. The example below illustrates this.

Example showing incompleteness of TWEAK-visit for MTC based termination: Suppose the domain consists of two operators: o_1 which has an effect p, and no preconditions; and o_2 which has an effect q and $\neg p$, and no preconditions. Suppose we start with an empty initial state, and want to achieve p and q in the goal state, and suppose if the goal p is considered for establishment first and then the goal q. TWEAK visit establishes p first and then q, then the only partial plan in the search space will have o_1 and o_2 unordered. The ground linearization $o_2 o_1$ is a solution, while $o_1 o_2$ is not a solution. Thus, MTC based solution constructor does not terminate on this plan, and TWEAK-Visit does not allow any further establishment refinements since both the goals have been considered for establishment once.

Of course, TWEAK-Visit could be made complete by replacing the MTC-based termination check with one that uses an implementation of *Some-Sol* (Definition 21).

5.2.4 The SNLP-CON, McNONLIN-CON Algorithms

SNLP-CON algorithm is similar to SNLP-MTC algorithm except that it does not use any tractability refinements, and uses a consistency check that explicitly checks for existence of safe ground linearizations (see Section 3.5). Specifically, the tractability refinement and consistency checks of SNLP-CON will be: [30]

2.b. Conflict resolution (Tractability Refinement): *None*.
3. Consistency check: Prune the partial plan if it has no safe ground linearizations.

In the same vein, McNONLIN-CON algorithm is similar to McNONLIN-MTC, except that it does not use any tractability refinements, and uses a consistency check that prune any partial plan that has no safe ground linearizations. [31]

[30] Perceptive readers might note a close relation between the operation of SNLP-CON and the idea of conflict deferment described in [32]. In particular, in comparison to SNLP and SNLP-MTC, SNLP-CON can be seen as deferring *all* threats. However, unlike the implementations of SNLP used in [32], SNLP-CON uses a full consistency check, and thus will never refine an inconsistent plan. See Section 9.

[31] There are several ways of implementing these full consistency checks. One is to enumerate all the ground linearizations, and check them one by one to see if any

Since SNLP-CON and McNONLIN-CON ignore tractability refinements, and do not consider a precondition for establishment more than once, as discussed in Sections 4.5.3 and 4.1, they are not guaranteed to be complete with respect to any *All-sol* solution constructor. As given here, they use MTC based termination (which is an implementation of *All-sol*), and thus could be incomplete. This can be illustrated by the same example we used to illustrate the incompleteness of TWEAK-Visit for MTC based termination check. Both SNLP-Con and McNONLIN-Con can of course be made complete by replacing the termination condition with one based on *Some-sol* (Definition 21).

6 Modeling and Analysis of Design Tradeoffs

We will start by developing two complementary models for the size of the search space explored by `Refine-Plan` in a breadth-first search regime. Recall, from Equation 1 in Section 2 that the search space size of a refinement search is related by the equation:

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

Search space size of any instantiation of `Refine-Plan` can be estimated with the help of the above equation. In the context of `Refine-Plan`, $\mathcal{K}$ is the candidate space of the problem (i.e., the candidate set of the initial null plan, $\mathcal{P}_\emptyset$ with which planning is initiated). b is the average branching factor of the search tree and d is the average depth of the search-tree. $\mathcal{F}_d$ is the d^{th} level fringe of the search tree explored by `Refine-Plan`. κ_d is the average size of the candidate sets of the partial plans in the d^{th} level fringe, and ρ_d is the redundancy factor, i.e., the average number of partial plans on the fringe whose candidate sets contain a given candidate in $\mathcal{K}$.

A minor technical problem in adapting this equation to planning is that according to Definition 15, both candidate space and candidate sets can have infinite cardinalities even for finite domains. However, if we restrict our attention to *planning domains with finite state spaces, solvable problems from those domains, and minimal solutions for those problems*, then it is possible to construct finite versions of both the candidate space and candidate set. Given a planning problem instance P, let l_m

---

of them are safe with respect to all auxiliary constraints. Another, which we used in our implementations of SNLP-CON and McNONLIN-CON, is to *simulate* the conflict resolution step on the given plan $\mathcal{P}$, and prune $\mathcal{P}$ if the conflict resolution step produces no refinements that are order and binding consistent. The difference between this use of conflict resolution and its normal use is that in the latter we replace $\mathcal{P}$ with the refinements generated by $\mathcal{P}$ (thereby increasing the branching factor).

be the length of the longest ground operator sequence that is a minimal solution of P (since the plan has a finite state space, all minimal solutions will be finite in length). Without loss of generality, 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.

If T_{rp} is the average per-invocation cost of `Refine-Plan` algorithm, then the total time-complexity of the `Refine-Plan` algorithm is:

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

We next analyze the complexity of `Refine-Plan` by fleshing out the parameters b, d and T_{rp}. 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.

Branching Factors: To expand the average branching factor b, we first define the following additional parameters. Let b_e be the number of successors generated by step 1.2. of `Refine-Plan`. This parameter is called the *establishment* branching factor, which can be further split into several parameters. We use b_{new} for the number of new operators found by step 1.2. for achieving $\langle c, s\rangle$, and b_{old} for the number of existing operators found by step 1 for achieving $\langle c, s\rangle$. Given a precondition $\langle c, s\rangle \in \mathcal{A}$ being established, for each establisher s' new or existing, step 1.2. makes sure that c persists from s' to s, by imposing additional constraints. The alternative choices in these constraints give rise to more successor plans. Let b_c be the number of successors generated corresponding to each new establisher s'. Then the establishment branching factor is

$$b_e = (b_{new} + b_{old}) \times b_c$$

Another contributor to the average branching factor is step 2, which applies additional refinements to make plan handling tractable. This step includes pre-ordering and conflict resolution, both of which involve imposing more constraints onto the plan. For each plan $\mathcal{P}$ generated by step 1, let b_t be the number of successor plans generated from $\mathcal{P}$ by step 2. b_t is called the tractability branching factor.

Putting both the components of the branching factor together, the average branching factor is

$$b = b_e \times b_t = (b_{old} + b_{new}) \times b_c \times b_t$$

Search Depth: Next, we consider the average search depth d. Let N be the length (in number of steps) of the solution, and P be the average number of preconditions per operator. Then, in the solution, there are $N \times P$ number of preconditions (tuples $\langle c, s \rangle$, where c is a precondition of the operator corresponding to step s). Let f be the fraction of the $N \times P$ pairs chosen by step 1.1. Let v be the total number of times any fixed pair $\langle c, s \rangle$ is chosen by step 1.1 (Note that v could be greater than one for planners that do not employ any bookkeeping steps.) Then we have

$$d = N \times P \times f \times v$$

Per-invocation cost of `Refine-Plan`: T_{rp} itself can be decomposed into four main components:

$$T_{rp} = T_{est} + T_{cons} + T_{tract} + T_{sol},$$

where T_{est}, is the establishment cost (including the cost of selecting the open goal to work on), T_{sol} is the cost of solution constructor, T_{tract} is the cost of tractability refinement, and T_{cons} is the cost of consistency check.

A summary of all the parameters used in the complexity model above can be found in Table 3. Armed with this model of the search space size and refinement cost, we will 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.

6.1 Tradeoffs offered by Solution Constructor

Stronger solution constructors allow the search to end earlier, reducing the average 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 T_{rp} (specifically the T_{sol} factor).

6.2 Tradeoffs offered by Goal Selection

Use of more sophisticated goal selection strategies increases the refinement cost (specifically the T_{est} factor). However, they can also bring about substantial reductions in the size of the explored search space size. For example, demand driven goal-selection strategies such as MTC-based goal selection take a least-commitment approach and establish a goal only when it is not necessarily true. This could either help in terminating before all goals are explicitly considered for establishment, or allowing the planner to work on them if and when they are no

$\|\mathcal{K}\|$	Size of the candidate space (i.e., size of the candidate set of ''null'' plan $\mathcal{P}_\emptyset$)
κ_d	Average size of the candidate sets of the nodes (partial plans) at d^{th}level fringe
ρ_d	Average number of times each candidate (ground op. seq.) occurs in the d^{th} level fringe
$\|\mathcal{F}_d\|$	Number of nodes (partial plans) at d^{th} level fringe
b	average *branching* factor
d	average search *depth*
b_{new}	average number of *new* establishers for a precondition
$bold$	average number of existing (or *old*) establishers for a precondition
b_e	average number of establishers for a precondition
b_t	successors after pre-ordering and conflict-resolution
b_c	successors generated by substeps (i) and (ii) in step 2.1
T_{rp}	per-invocation cost of `Refine-Plan`
T_{est}	cost of establishment refinement
T_{tract}	cost of tractability refinement
T_{cons}	cost of consistency check
T_{sol}	cost of solution constructor (termination check)
N	total number of operators in a plan
P	total number of *preconditions* per operator
f	*fraction* of the preconditions of the plan ($\langle c, s\rangle$) considered by the establishment refinement
v	average number of times a precondition ($\langle c, s\rangle$) is considered by the establishment refinement (*visited*)

Table 3
Summary of the parameters used in describing the complexity of `Refine-Plan`

longer necessarily true. ater stages of the search. Either way, this could reduce the average establishment branching factor b_e, the average depth and consequently the search space size. [32]

[32] It is important to note the difference between demand-driven goal selection and modal truth criterion. Since modal truth criterion becomes NP-hard for plans containing actions with conditional effects, it looks as if demand driven goal selection also becomes intractable for such plans. This is not the case since one does not have to use a *necessary and sufficient* modal truth criterion for implementing demand-driven goal selection. Since the order of

6.3 Tradeoffs offered by Bookkeeping

Addition of bookkeeping techniques tend to reduce the redundancy factor ρ_d, and the average candidate set size κ_d (since fewer ground linearizations will be safe with the added auxiliary constraints). In particular, as we observed earlier, use of contributor protections makes the search *systematic*, eliminating all the redundancy in the search space and making ρ_d equal to 1 [24,11]. This tends to reduce the fringe size, $|\mathcal{F}_d|$. Bookkeeping 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. [35]). In terms of the b-d view of the fringe size, use of bookkeeping techniques tends to reduce the branching factor (assuming the optional consistency check is being used) while keeping the average depth of the search constant.

6.4 Tradeoffs offered by 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* (Defn. 7) In terms of the refinement cost, stronger consistency checks tend to be costlier, thereby driving up the refinement cost (in particular T_{sol}). As mentioned earlier, sound and complete consistency check is NP-hard even for ground plans, if the plans contain interval preservation constraints.

6.5 Tradeoffs offered by 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 T_{cons} 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

goal selection does not affect completeness, any tractable approximation to the modal truth criterion will be enough.

T_{sol} and T_{est} components) [11]. 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.

7 Predicting performance

In the previous section, we discussed the way design choices in instantiating individual steps of the `Refine-Plan` algorithm affect the search space size and the refinement cost. An important insight given by this discussion is that most individual design choices (such as goal selection, termination, bookkeeping, tractability refinement) can be seen as trading the complexity between search space size and refinement cost. As such, we will not expect any universally valid dominance results among them. Rather, the performance will depend upon domain-dependent factors such as solution density.

Further, the performance depends on the overall effect of the tradeoffs provided by the specific design choices. While the discussion above shows how individual design choices affect the size of the search space size in isolation, it does not account for the interaction between the choices made in two separate steps. Given this, an obvious question that arises is: *Can we predict which instantiation of* `Refine-Plan` *will perform the best in a given domain*. Of particular interest will be any predictions that are made in terms of easily measurable features of the domain. This is the question that we shall address in this section.

Given the many possible dimensions of variation of `Refine-Plan`, we would like to focus our attention on those dimensions that account for the prominent differences between existing planners. From our discussion in Section 5 (as well as the characterization of the various planners in Table 1), we note that two of the most prominent differences among the various planning strategies are the specific bookkeeping strategies employed by them, and the specific types of tractability refinements they use. In this section, we will attempt to predict the effect of these differences on the performance on practical problems. To make our exposition simpler, we will restrict our attention to propositional planning domains.

7.1 Tractability Refinements

Our discussion about design tradeoffs shows that tractability refinements aim to reduce the cost of consistency check by increasing the search space size. The more eager the tractability refinement, the larger the b_t factor, and the larger the search space increase. For the case of breadth-first search regimes, with other factors of b kept constant, the increase in b_t will increase the search space size exponentially. Although tractability refinements aim to reduce the cost of the consistency check, unless this relative reduction is also exponential, we will not expect stronger tractability refinements to improve performance.

However, under certain circumstances, it is possible for tractability refinements to have interactions with other parts of the planner. In particular, while increase in b_t should in general increase b, the additional linearization of the plan done at the tractability refinement stage may sometimes wind up reducing the number of establishment refinements for the conditions considered in the later stages. In particular, when the plan is more linear, there are fewer existing steps that can act as establishers for a condition. This reduces b_{old}. Furthermore, when the plan is more linear, there are fewer steps in the plan that can violate an establishment. Thus the number of declobbering choices for each establishment (i.e., the number of ways of preserving the precondition that is just established) will also be lower. This reduces b_c. Since b_e is proportional to the product of b_{old} and b_c, it also reduces b_e. The more eager the tractability refinement, the more linear the partial plan, and the higher the reduction in b_{old}.

The reduction in b_e could, sometimes, offset the increase in b_t, reducing the overall branching factor in the presence of tractability refinements. Whether or not this will happen in practice depends on the features of the domain, as well as the specific order in which goals are considered for establishment during planning. In particular, two (static) characteristics of a domain that are relevant are: (i) the average number of actions that can establish a precondition or top-level goal and (ii) the average number of actions that can delete a precondition or top-level goal, in a partial plan. We will call these two factors $\#_{est}$ and $\#_{clob}$ respectively.

We will call a goal condition or a precondition c a **high-frequency condition** if it has very high $\#_{est}$ and $\#_{clob}$ factors. Since high-frequency conditions have many potential establishers and clobberers, establishment refinement on such conditions has potentially high b_{old} and b_c factors, and can thus benefit from the interaction between b_t and b_e. Moreover, when the domain contains only a few high-frequency conditions, the order in which high-frequency conditions are established relative to the other preconditions will guide the interaction between the b_t and b_e factors.

In summary, given two instantiations I_e and I_c of `Refine-Plan` that differ only in the type of tractability refinements they employ, such that I_e uses a more eager tractability refinement than I_c (i.e., b_t factor for I_e is greater than that of I_c), we can make the following predictions about the relative performance of I_c and I_e on a population of problems from a domain D.

Hypothesis 1 *If none of the conditions in the domain are high-frequency conditions, then there will be no interaction between b_t and b_e, and since I_c has a lower b_t than I_e, it will have a smaller search space than I_e. It will also have a lower time-complexity as long as the per-node cost of I_e is not exponentially lower than that of I_c.*

Hypothesis 2 *If all (or most) of the conditions in the domain are high-frequency conditions (i.e., $\#_{est} \gg 1$ and $\#_{clob} \gg 1$ for all conditions), then I_e may have smaller search space than I_c because of the interaction between b_t and b_e. Thus, I_e could perform better than I_c.*

Hypothesis 3 *If only some of the conditions in the domain are high frequency conditions, and the goal-selection strategy is such that the high-frequency conditions are considered for establishment earlier, then I_e may once again have a smaller search space than I_c, and thus could perform better than I_c.*

7.2 Bookkeeping(Protection) Strategies

Bookkeeping strategies by themselves do not increase the search space size, as they do not contribute to the increase of the branching factor. The difference between a planner employing a bookkeeping strategy and one that does not employ a bookkeeping strategy, is that the partial plans in the former have more constraints than those of the latter. Thus, the partial plans in the former can become inconsistent earlier than those in the latter. This can lead to increased backtracking. The flip side is that the size of the overall search space is smaller for planners with protection strategies than for planners without protection strategies (as bookkeeping constraints tend to reduce redundancy in the search space).

Thus, protection strategies have two possible effects on the size of the explored search space [13]:

- They reduce the redundancy in the search-space and thus make the overall search space smaller.
- They represent higher commitment to establishment choices, and thus can lead to higher backtracking during search.

The combined effect of these two factors on the size of the explored search space depends on the type of search strategy used by the planner, the type of conditions in the domain, as well as the *apparent solution density* of the planning problem.

For problems with low solution densities, the planners will be forced to look at a large fraction of their search space, and thus the size of the overall search space will have an effect on the performance of the planner. In particular, planners using strong protection strategies will ensure smaller overall search spaces, and thus lower planning cost.

For problems with high solution densities, the size of the overall search space has no appreciable correlation with the size of the explored search space, and thus we do not expect redundancy reduction afforded by protection strategies to lead to improved performance. On the other hand, the increased commitment to establishment choices entailed by protection strategies can lead to increased backtracking (see [12,13]), which can degrade performance (this effect is more apparent in depth-first search regimes).

Solution density has a lower correlation with performance in the case of breadth-first search regimes, which search all branches of the search space. For these, the effect of protection strategies will be felt in terms of the branching factor reduction. Specifically, planners with stronger protection strategies will prune more partial plans and will thus have smaller average branching factor than weaker protection strategies. This should improve performance, unless the cost of maintaining consistency of the book-keeping constraints posted by stronger protection strategies is high.

The protection strategies also have interactions with the other steps of the `Refine-Plan` algorithm, and their effect on performance can sometimes be modified by the combination. For example, the harmful effects of increased commitment to specific establishers can be overcome to some extent by using goal-selection strategies such as those based on modal truth criterion which take a demand-driven approach to goal establishment (by working only on goals that are not necessarily true), or by using precondition abstraction strategies that work on most-constrained goals (i.e., goals with fewest establishers) first. In both cases, there is a smaller amount of branching in the search tree, and thus the chances of committing to a wrong establishment choice are lower.

Similarly, although protection strategies themselves do not increase the search space size, they do indirectly determine the amount of refinement done at the tractability refinement step. For example, planners using stronger protection strategies will be posting more bookkeeping constraints, and thus have to do more pre-ordering or conflict resolution refinements, leading to higher branching factors. (It is important to note however that this increase in search space size is not a necessary side-effect of protection strategies since tractability refinements are an optional step in `Refine-Plan`.)

Let us summarize, and restrict our attention to the first-order effect of protection strategies on performance: given two instances of `Refine-Plan`, I_{sp} and I_{np}, which differ only in their use of bookkeeping strategies, such that I_{sp} uses stronger protections compared to I_{np}, we can make the following predictions on the relative performance of I_{sp} and I_{np} on a populations of problems from a domain D:

Hypothesis 4 *If the domain has a high solution density, then there will be no appreciable difference in the relative performance of I_{sp} and I_{np}, unless there is a high probability that the planner will commit to wrong establishments in the beginning of the search (this could happen, for example, when the domain contains many high-frequency conditions [13]).*

Hypothesis 5 *If the domain has a low solution density, the planners are forced to explore a large part of their search space. Thus, the redundancy in the search space will become a factor affecting the performance of the planner. Specifically, the size of the search space explored by I_{sp} could be smaller than that of I_{np}. Thus, I_{sp} could perform better than I_{np}.*

Hypothesis 6 *For breadth-first search regimes, stronger protection strategies reduce the overall branching factor and can reduce the average size of the search space size explored. Thus, I_{sp} can once again perform better than I_{np}.*

8 Empirical Evaluation of Performance Predictions

In this section we will discuss the results of a series of empirical studies with several instantiations of `Refine-Plan`. The aim of the empirical study is two-fold:

-- To provide experimental support for our hypotheses regarding the effect of tractability refinements and bookkeeping constraints
-- To demonstrate that the complexity model developed in the previous section helps in explaining the empirical performance.

The following sections briefly describe the planning domains that we used in our experiments, and the experiments themselves. Before we go into those details however, a few comments about the experimental methodology are in order. All the planners used in our experiments are described as instantiations of `Refine-Plan` in Section 5 (and summarized in Table 1). We implemented them all as instantiations of `Refine-Plan` algorithm, thereby allowing sharing of many common routines, and facilitating a fair comparison. Moreover, when evaluating the hypotheses about a specific component of `Refine-Plan`, we made sure to compare only the planners that correspond to instances of `Refine-Plan` that are equivalent in all other components. This type of normalization was a direct result of the unified view of partial order planners provided by `Refine-Plan`.

In many of the experiments, the planners were tested on random population of problems from several artificial domains. Unless otherwise stated, all the experiments used a breadth-first search. To account for extraneous variations, all the plots were made with averages over 25 runs. All the experiments were run with a per-problem CPU time limit of 60 seconds. This meant that in some runs the planners were aborted before solving the problem. In all cases, we have provided plots showing the number of unsolved problems corresponding to each data point. Since the final averages were taken over both solved and unsolved problems in a set, some of the averages underestimate the actual numbers for those planners that failed to solve many problems within the resource limit. This point should be kept in mind in viewing the various performance plots presented in this section.

Finally, most of the experiments described in the following sections have been validated by running them on two largely independent implementations (see [16] and [21]). This lends us confidence that the results are not dependent on any idiosyncrasies of a particular implementation. The final data reported in this paper correspond to runs on a SUN SPARC 20 with 96 megabytes of main memory running Lucid Commonlisp. The code and the data for replicating our experiments as well as all the raw data collected from the experiments are available via the Internet. [33]

8.1 Experimental Domains

In our experiments, we concentrated on artificial domains as they provide for a controlled setting to evaluate our hypotheses about performance tradeoffs. Specifically, we experimented with two families of artificial domains.

The first family of domains that we considered is the the ART-$\#_{est}$-$\#_{clob}$ family. These domains were designed to provide a way of controlling for the $\#_{est}$ and $\#_{clob}$ factors of the preconditions and goals in the domain. Specifically, 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 to this basic structure, we add extra operator effects to change the $\#_{est}$ and $\#_{clob}$ factors of the conditions.

An example of an operator schema from ART-$\#_{est}$-$\#_{clob}$ domain is shown below. Let n be the total number of goals to be achieved. Let the goal state be $G = \{G_i, i = 0, 1, \ldots, n-1\}$. Assume that the initial state I is $\{I_i, i = 0, 1, \ldots, n-1\}$. Each goal G_i can be achieved by operator A_{i2}, and the precondition P_i for A_{i2} can be achieved by operator A_{i1}. The average $\#_{est}$ factor is controlled by a number n_+ (where $n_+ < n$), and the average $\#_{clob}$ factor by a number n_- (where $n_- < n$). For $\#_{est}$, it is assumed that for the first n_+ operators in a solution plan, each operator

[33] From ``ftp://ftp.isi.edu/sims/code/refine-plan.tar.Z''.

achieves a precondition of an operator for the next subgoal. In a similar manner, for $\#_{clob}$ only the first n_- operators interact, and each operator deletes a precondition of another operator for a previous subgoal.

(defstep
    :action A_{i1}
    :precond I_i
    :add $\{P_i\} \bigcup \{I_{i+1}$ if $i < n_+\}$
    :equals $\{\}$
    :delete $\{I_{i-1}$, if $0 < i < n_-\}$
)
(defstep
    :action A_{i2}
    :precond P_i
    :add $\{G_i\} \bigcup \{P_{i+1}$ if $i < n_+\}$
    :equals $\{\}$
    :delete $\{P_{i-1}$, if $0 < i < n_-\}$
)

The second domain that we experimented with is called ART-MD-RD is given below:

For even i:
(defstep
    :action A_i
    :precond I_i, *he*
    :add G_i, *hf*
    :equals $\{\}$
    :delete $\{I_j | j < i\} \cup \{he\}$)
For odd i:
(defstep
    :action A_i
    :precond I_i, *hf*
    :add G_i, *he*
    :equals $\{\}$
    :delete $\{I_j | j < i\} \cup \{hf\}$)

Unlike the ART-$\#_{est}$-$\#_{clob}$ domains, where all the conditions have the roughly same average $\#_{est}$ and $\#_{clob}$ factors, ART-MD-RD domain contains two high-frequency conditions, e.g. hf and he, with the rest being low-frequency conditions (e.g. I_i and G_i). Thus, the order in which the goals are established has a significant impact on the complexity of the search as well as the relative performance of the various planners in ART-MD-RD. In particular, the performance of a planner depends to a large extent on *whether* and *when* the planner explicitly considers hf and he for establishment.

A third domain that we used in some of our experiments is ART-MD, which is a variant of ART-MD-RD, where the actions do not have hf and he in their preconditions and effects. (ART-MD is identical to the D^1S^1 domain used in Barret et al.'s experiments [2].)

8.2 Evaluation of Hypotheses about Tractability refinements

To evaluate our hypotheses about the effect of tractability refinements on performance (see Hypotheses 1, 2 and 3 in Section 7.1), we conducted two separate experiments: one with the problems from ART-$\#_{est}$-$\#_{clob}$ domain and the other with the problems from ART-MD-RD domain. The first one helps us evaluate our hypotheses in domains where all the preconditions have uniform $\#_{est}$ and $\#_{clob}$ factors, while the second one helps us evaluate our hypotheses in domains where the preconditions have non-uniform $\#_{est}$ and $\#_{clob}$ factors. We describe both in turn.

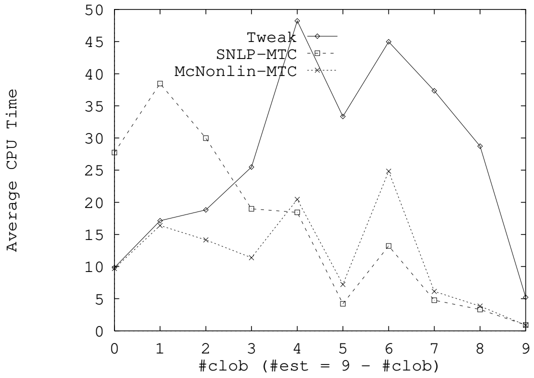
8.2.1 Experiments in ART-$\#_{est}$-$\#_{clob}$ domain

In our first experiment, we used problems from ART-$\#_{est}$-$\#_{clob}$ domain and simultaneously varied the $\#_{est}$ and $\#_{clob}$ factors, such that the $\#_{clob}$ factor increased from 0 to 9 and the $\#_{est}$ factor decreased from 9 to 0, while the sum of the two factors remained a constant 9. The number of goals in each problem instance was 10. We compared the relative performance of three planners -- TWEAK, SNLP-MTC and McNONLIN-MTC -- which use three different tractability refinement strategies. For any pair of ($\#_{est}, \#_{clob}$) values, we ran the planners on 25 randomly generated problems. The plots in Figures 9 and 10 show the averages of various features of the search space and refinement cost model.

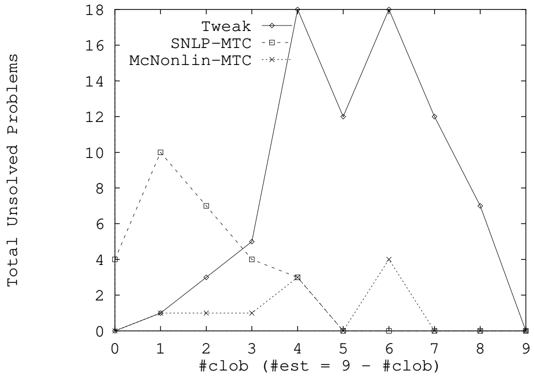
From Section 5, we know that TWEAK does not do any tractability refinement (thus, its $b_t = 1$) while SNLP-MTC and McNONLIN-MTC do conflict resolution based tractability refinement. We also know that SNLP-MTC which does conflict resolution with respect to contributor protections, corresponds to a more eager tractability refinement strategy than McNONLIN-MTC which does conflict resolution with respect to interval protection. Thus, we have:

b_t(SNLP-MTC) $\geq$ b_t(McNONLIN-MTC) $\geq$ b_t(TWEAK)

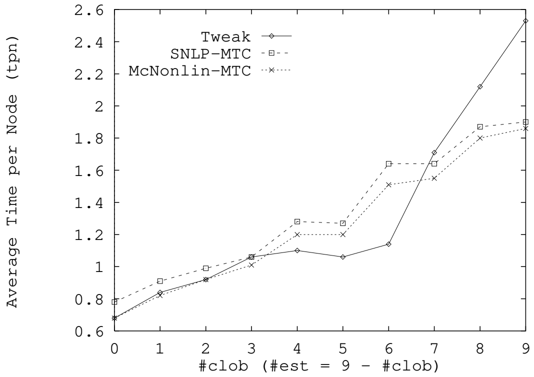
Based on Hypothesis 2 regarding tractability refinement, we expect that SNLP-MTC and McNONLIN-MTC should perform better than TWEAK when the domain contains high-frequency conditions (i.e., conditions with high $\#_{est}$ and $\#_{clob}$ factors). With respect to our current experiment in ART-$\#_{est}$-$\#_{clob}$ domain, this happens when



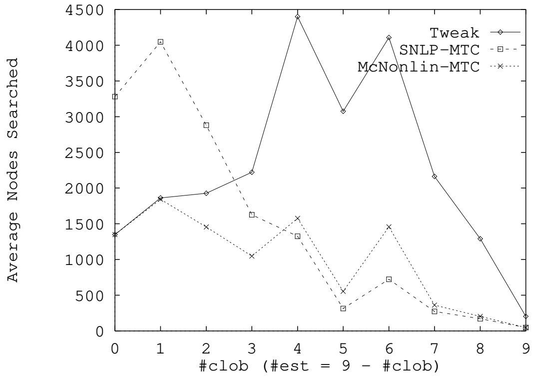
(a) Average CPU time



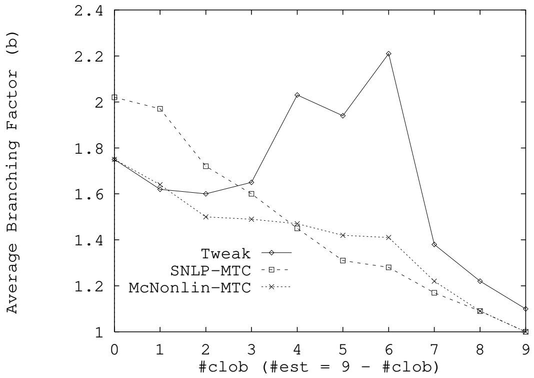
(b) # Unsolved Probs.



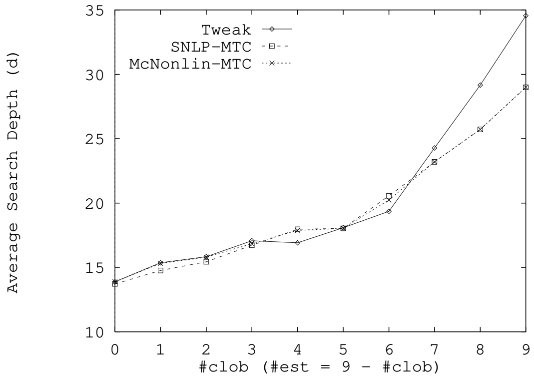
(c) Avg. T_{rp}



(d) Avg. # nodes

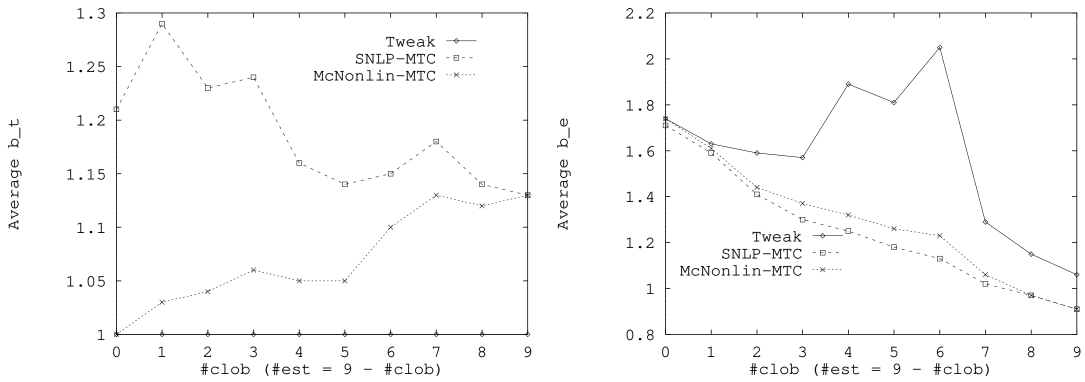


(e) Avg. b



(f) Avg. d

Fig. 9. Plots comparing relative performance of normalized planners in ART-$\#_{est}$-$\#_{clob}$ domain, where $\#_{est} + \#_{clob}$ is kept constant at 9.



(a) Avg. b_t (b) Avg.b_e

Fig. 10. Analyzing the branching factors of the planners in ART-$\#_{est}$-$\#_{clob}$ domain, where $\#_{est} + \#_{clob}$ is kept constant at 9.

$\#_{est} \approx \#_{clob}$. Since we are keeping $\#_{est} + \#_{clob}$ constant at 9, we expect that this corresponds to the region in the middle of the graph around $\#_{clob} = 4$. In these instances, we expect that the planners SNLP-MTC and McNONLIN-MTC, which are eager to use tractability refinements, will perform better than TWEAK, which uses no tractability refinements.

Figure 9 compares a variety of factors of the search space size and refinement cost for TWEAK, SNLP-MTC and McNONLIN-MTC. From the plots of average CPU time and number of unsolved problems (Figure 9(a) and Figure 9(b) respectively), we see that TWEAK's performance deteriorates significantly around points with $\#_{est} \approx \#_{clob} \approx 4$. We also note that around these middle points both SNLP-MTC and McNONLIN-MTC perform much better than TWEAK. This conforms to Hypothesis 2. Using a signed-rank test [7] on cpu times, these results are statistically significant at $\#_{clob} = 4$ with p-values of 0.01 and 0.008, [34] respectively.

To show that this observed performance differential is correlated with the interplay between b_t and b_e factors, we compare the average b_t and b_e factors of the three planners in Figure 10. We note that although b_t is higher for McNONLIN-MTC and SNLP-MTC as expected, the b_e factor varies according to our hypothesis. In particular, we note that around the region of $\#_{clob} \approx 4$, where all the conditions in the domain become high-frequency conditions, TWEAK's b_e attains its maximum. At the same time, SNLP-MTC and McNONLIN-MTC benefit from the additional linearization provided by their eager tractability refinements, and thus show significantly lower b_e (and consequently have lower b, despite higher b_t).

[34] The signed rank test generates an upper bound on what is called the p-value. The p-value is the probability that conclusions drawn from the data are in error. The lower the p-value, the stronger the evidence that the hypotheses are correct. In all of these comparisons the significance level is taken to be 0.05. When the p-value is below the significance level the results are considered to be statistically significant.

Thus, this experiment supports our prediction that tractability refinements are useful when *both* the establishment and tractability branching factors are high enough. One might still ask *what would happen if only one of* $\#_{est}$ *and* $\#_{clob}$ *is high and the other one is low.* Since there are no high-frequency conditions in such domains, according to Hypothesis 1, we would not expect any interplay between b_t and b_e and thus the performance of SNLP-MTC and McNONLIN-MTC should be either worse than or equal to that of TWEAK.

The plots in 9(a) 9(b) validate this prediction. At both extreme points when only one of $\#_{est}$ and $\#_{clob}$ is high, planners with strong tractability constraints do not have any notable performance advantage. At both ends McNONLIN-MTC performs similarly to TWEAK, and when $\#_{est}$ is largest SNLP-MTC performs much worse than TWEAK. These phenomena fit exactly with our prediction. When $\#_{clob} = 9$ and $\#_{est} = 0$ the tractability constraints do not have any additional effect on the establishment branching factor b_e, since b_e is already at its lowest value. Thus, all three planners have the same performance. On the other hand, when $\#_{est} = 9$ and $\#_{clob} = 0$, SNLP-MTC generates more refinements in each step than TWEAK due to its more eager tractability refinement strategy, but the reduction in its establishment branching factor does not justify the increase in tractability branching factor (see plots in Figure 10). Thus, at this point, SNLP-MTC performs worse than TWEAK.

Importance of Normalization: One might ask why we used the non-standard planners SNLP-MTC and McNONLIN-MTC, instead of SNLP and McNONLIN. The obvious reason of course is that seen as instances of `Refine-Plan`, SNLP and McNONLIN differ from TWEAK in aspects other than tractability refinements. Let us elaborate on this further by demonstrating the importance of using the normalized planners. In Figure 11, we provide the comparisons between the brand-name planners SNLP, McNONLIN, TWEAK and UA, instead of the normalized instances we used in the previous experiments. Note that unlike SNLP-MTC and McNONLIN-MTC, SNLP and McNONLIN do not outperform TWEAK in the region around $\#_{est} \approx \#_{clob} \approx 4$. In fact, we find that SNLP and McNONLIN perform worse or about the same as TWEAK for most of the points. This discrepancy is easy to explain once we note that SNLP and McNONLIN differ from TWEAK not only in tractability refinements, but also in the goal-selection and termination criteria they use. In particular, SNLP and McNONLIN insist on working on each precondition exactly once, while TWEAK uses MTC and takes the more demand-driven approach -- working only on those goals that are not necessarily true. This difference winds up drowning the effect of tractability refinements. In fact, we can see from Figure 11(d) that the fraction of preconditions visited by SNLP and McNONLIN is considerably higher than that of TWEAK in the middle range.[35] Consequently, the solution depth is also higher, and reduction in b due to the

[35] The f values of SNLP and McNONLIN are depressed towards the beginning because they fail to solve most of the problems in that region.

$b_t - b_e$ interaction is not enough to offset this. The use of different goal-selection and termination strategies also affects the per-invocation cost of `Refine-Plan`. The plots in Figure 11(c) show that TWEAK has a higher T_{rp} than SNLP and McNONLIN.

In contrast, once we normalize the goal-selection and termination criteria, the fraction of preconditions visited by TWEAK, SNLP-MTC and McNONLIN-MTC get much closer (See Figure 12(a)), thus allowing the effect of tractability refinements to stand out. (The stark performance difference between SNLP and SNLP-MTC, as well as McNONLIN and McNONLIN-MTC, also demonstrate the utility of demand-driven goal selection strategies.)

8.2.2 Experiments in ART-MD-RD domain

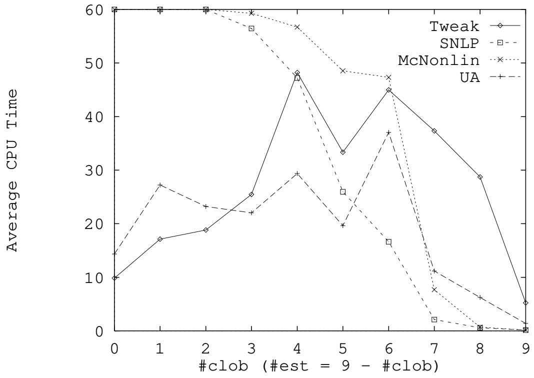
All the ART-$\#_{est}$-$\#_{clob}$ domains are uniform in that in each of them all the preconditions have roughly the same number of $\#_{est}$ and $\#_{clob}$ factors. To evaluate Hypothesis 3 regarding the domains that have precondition with differing $\#_{est}$ and $\#_{clob}$ factors, we experimented with problems from ART-MD-RD domain. In addition to the three planners used in ART-$\#_{est}$-$\#_{clob}$ domain, we also included two other planners, UA and SNLP-UA, which use pre-ordering based tractability refinements (see Section 5). These two represent tractability refinement strategies that are even more eager than those of SNLP-MTC and McNONLIN-MTC. Specifically, the b_t factors of these planners are expected to follow the order:

b_t(SNLP-UA) $\geq$ b_t(UA) $\geq$ b_t(SNLP-MTC) $\geq$ b_t(McNONLIN-MTC) $\geq$ b_t(TWEAK).

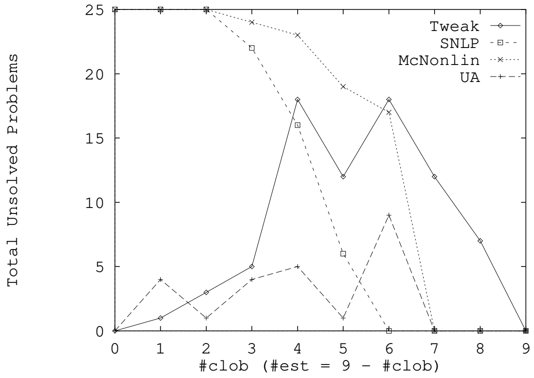
Figure 13 shows the performance of the five planners, SNLP-MTC, McNONLIN-MTC, UA, TWEAK and SNLP-UA on problems from ART-MD-RD domain. Each point in the plot corresponds to an average over 25 random problems with the given number of goals (drawn from $\{G_1 \cdots G_8\}$). The initial state is the same for all the problems, and contains $\{I_1 \cdots I_8\} + \{he\}$.

Recall that Hypothesis 3 in Section 7.1 predicts that the relative performance of the planners in such situations will depend upon the order in which the high-frequency conditions are established relative to the low-frequency ones. We thus experimented with two different goal-selection 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.

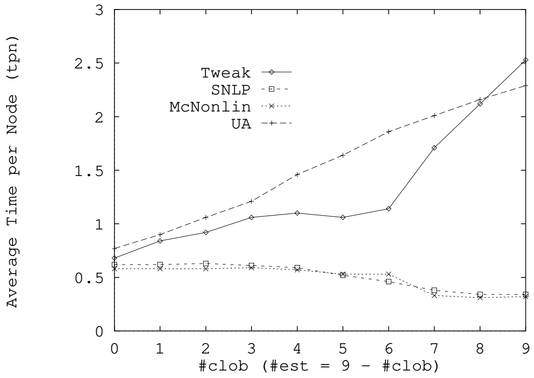
From the description of ART-MD-RD domain in Section 8.1, it is easy to see that hf/he are high-frequency conditions, with $\#_{est} \approx \#_{clob} \approx 4$, while G_i and I_i are



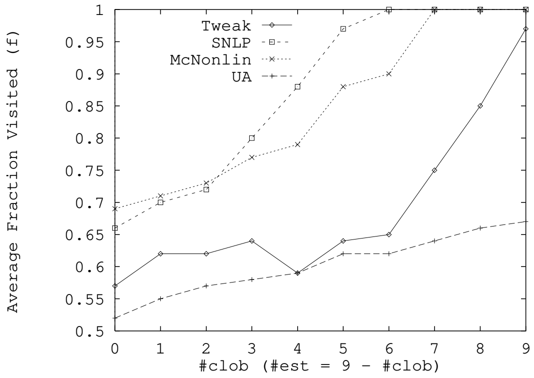
(a) Average CPU time



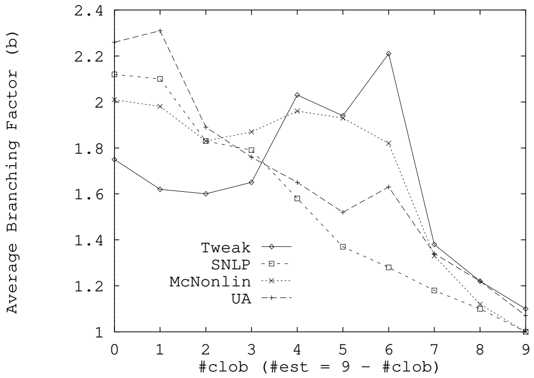
(b) # Unsolved Probs.



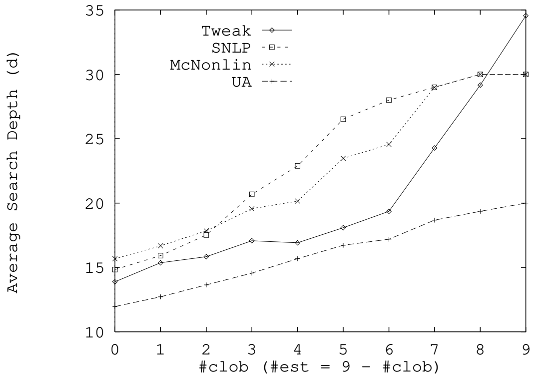
(c) Avg. T_{rp}



(d) Avg. f

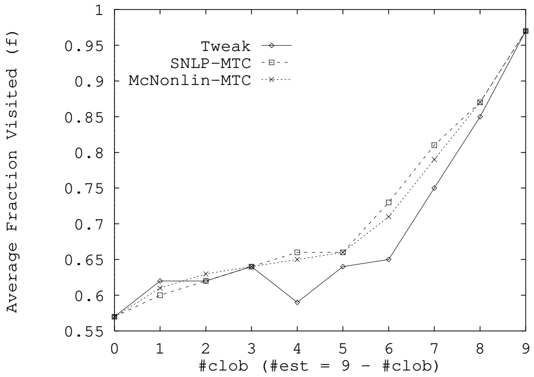


(e) Avg. b

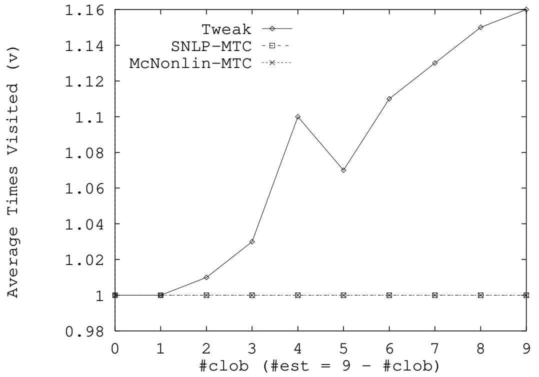


(f) Avg. d

Fig. 11. Plots comparing relative performance of existing (brand-name) planners in ART-$\#_{est}$-$\#_{clob}$ domain, where $\#_{est} + \#_{clob}$ is kept constant at 9.

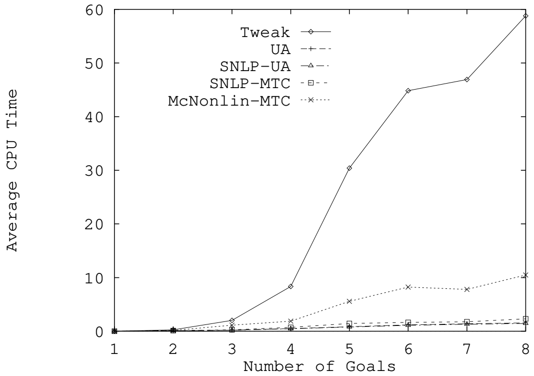


(a) Avg. f

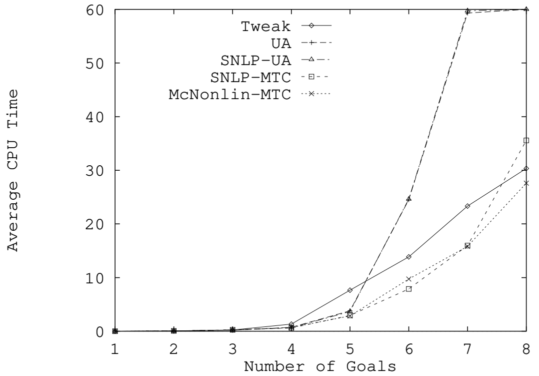


(b) Avg. v

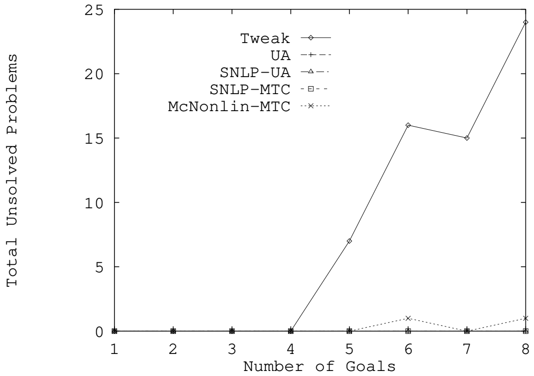
Fig. 12. Analyzing the solution depths of the planners in ART-$\#_{est}$-$\#_{clob}$ domain.



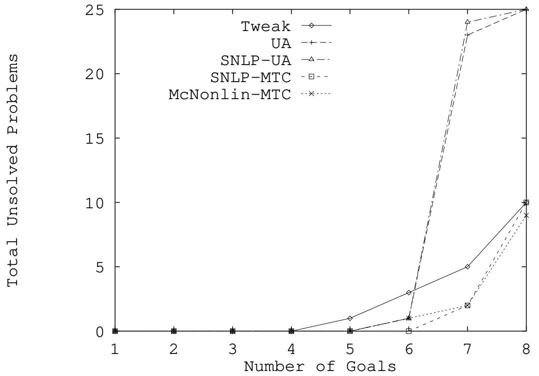
(a) Avg. CPU time (LIFO)



(b) Avg. CPU time (FIFO)

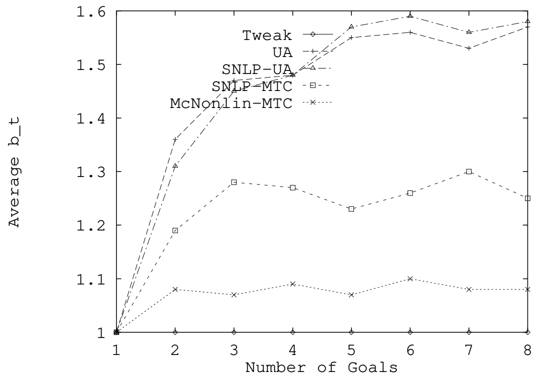


(c) # Unsolved (LIFO)

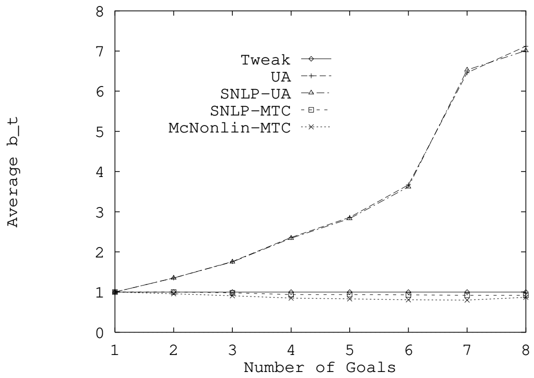


(d) # Unsolved (FIFO)

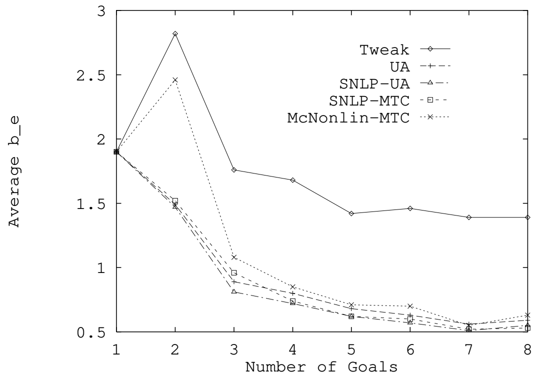
Fig. 13. Comparative performance in ART-MD-RD domain



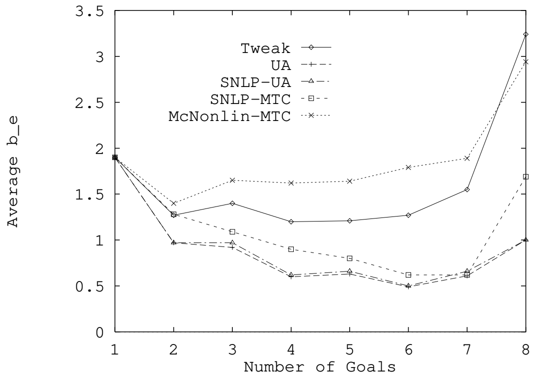
(a) Avg. b_t (LIFO)



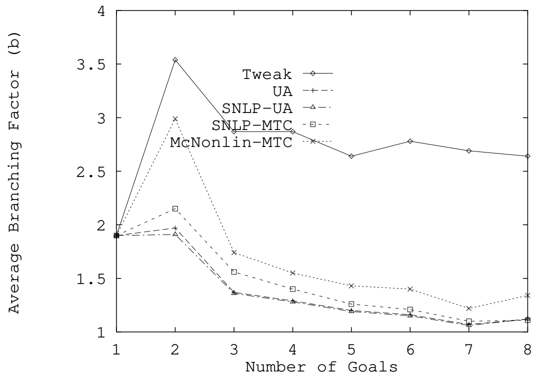
(b) Avg. b_t (FIFO)



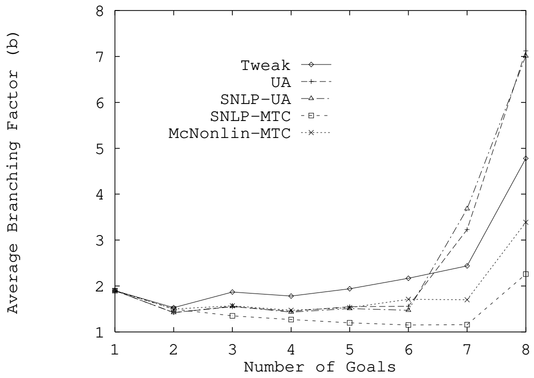
(c) Avg. b_e (LIFO)



(d) Avg. b_e (FIFO)



(e) Avg. Branching Factor(LIFO)



(f) Avg. Branching Factor(FIFO)

Fig. 14. Plots comparing average branching factors in ART-MD-RD experiments.

low frequency conditions (with $\#_{est} \approx 1$ and $\#_{clob} \approx 8$). Looking at the description of ART-MD-RD, we also note that in LIFO ordering, hf and he are considered for establishment in the early parts of the search, while in FIFO ordering, they are considered after all G_i are considered for expansion. Further, in FIFO ordering, establishing I_i and G_i linearizes the plan and consequently implicitly establishes the hf/he preconditions of the actions. Thus, under MTC-based goal-selection strategies that are used by the five planners, hf/he will rarely be considered for expansion. Based on this reasoning, and our hypotheses regarding the effect of tractability refinements on performance, we would expect that the planners using more eager tractability refinements will perform better than the planners using less eager tractability refinements in LIFO ordering. Similarly, in FIFO ordering, we would expect the planners using eager tractability refinements to perform worse.

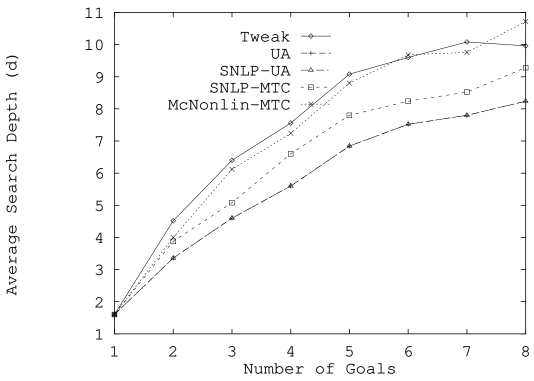
Figure 13 compares the performance of the five planners on problems in ART-MD-RD domain for both LIFO and FIFO. We start by noting that our predictions regarding relative performance are borne out by the plots comparing average CPU time and the number of unsolved problems of the five planners across the two goal orderings. Specifically, we note that UA and SNLP-UA, which use the most eager tractability refinements, outperform TWEAK in LIFO, but perform worse than TWEAK in FIFO. These comparisons are statistically significant for the average CPU time over the entire data set with p-values of 0 (using signed rank test).

Figure 14 compares the average tractability branching factor b_t, establishment branching factor b_e, and the overall branching factor b across the five planners and two goal selection strategies. These plots show that the performance of the planners is correlated with the b_t - b_e interaction as per our hypothesis. Specifically, from the average b_t plots in Figure 14(a) and Figure 14(b), we note that relative values of b_t across both goal-orderings are in accordance with our expectation:

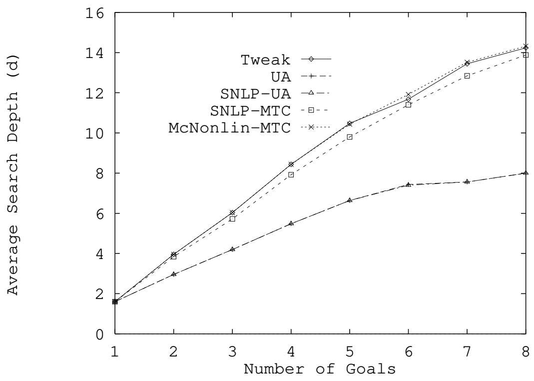
b_t(SNLP-UA) $\geq$ b_t(UA) $\geq$ b_t(SNLP-MTC) $\geq$ b_t(McNONLIN-MTC) $\geq$ b_t(TWEAK).

The overall branching factors, shown in Figures 14(e) and 14(f) follow the relative pattern of b_t in the case of FIFO ordering, but are opposed to the relative pattern of b_t in the case of LIFO ordering. We note that the relative plots of the overall branching factors are fully correlated with the average CPU time plots in Figures 13(a) and 13(b).

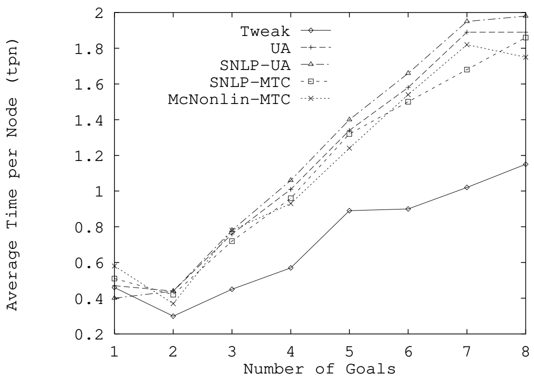
The plots in 14(c) and 14(d) show the average establishment branching factors of the five planners in LIFO and FIFO goal orderings. From these, we also note that the relative pattern of the overall branching factors is fully correlated with our hypothesis regarding the $b_t - b_e$ interaction. Specifically, in the case of LIFO ordering, which explicitly considers the high-frequency hf/he conditions for establishment, the b_e values are much lower for planners using eager tractability refinements. Since $b = b_t \times b_e$, the overall branching factor of planners with eager



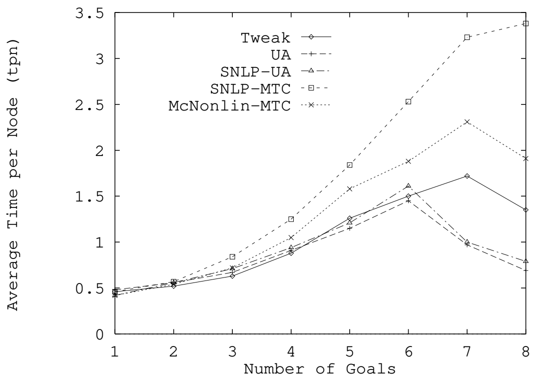
(a) Avg. soln. depth. (LIFO)



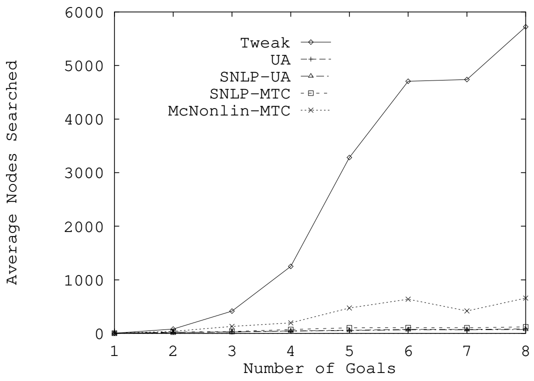
(b) Avg. soln. depth (FIFO)



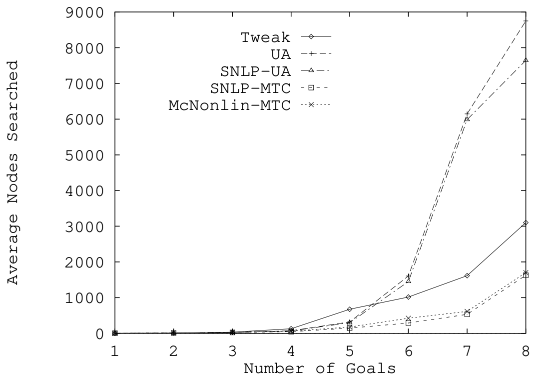
(c) Avg. per node cost T_{rp} (LIFO)



(d) Avg. per node cost T_{rp} (FIFO)



(e) Avg. search space size $\mathcal{F}_d$ (LIFO)



(f) Avg. search space size $\mathcal{F}_d$ (FIFO)

Fig. 15. Plots comparing solution depth, per-node cost and search space size in ART-MD-RD experiments.

tractability refinements (and higher b_t)is lower. In the case of FIFO, the increase in b_t is not offset by the reduction in b_e. Consequently, tractability refinements do not provide any performance advantages. Instead, the additional branching introduced by the planners with most eager tractability refinement strategies, UA and SNLP-UA, winds up increasing their overall branching factor, thereby significantly degrading the performance.

From the plots of per-node cost of refinement, solution depth and explored search space size in in Figure 15, we note that there are no other dominant competing explanations for the observed performance differentials. Specifically, although there are minor differences in the relative values of per-node cost (Figures 15(c) and 15(d)) and solution depth (Figures 15(a) and 15(b)) across the planners, the plots of explored search space size (Figures 15(e) and 15(f)) follow the same pattern as the overall branching factor and the average CPU time.

Finally, our experiments also show that planners using conflict resolution strategies (SNLP-MTC and McNONLIN-MTC) strike a middle-ground in terms of performance, across both goal orderings. Since the orderings introduced by them are more sensitive to the role played by the various steps in the plan, they seem to avoid both the excessive b_t of planners using preordering strategies (UA, SNLP-UA), and the excessive b_e of planners not using any form of tractability refinement (TWEAK).

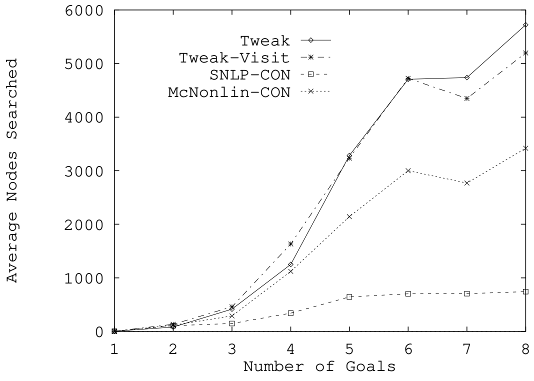
8.3 Evaluation of Hypotheses regarding Bookkeeping strategies

To evaluate our hypotheses regarding the correlation between the solution density and the effect of strong protection strategies, we ran two different experiments. In the first experiment, we used problems from ART-MD-RD domain described earlier. In the second we used problems from ART-MD and randomly misdirected each planner with respect to a *min-goals* heuristic. We will describe the these experiments and their results below.

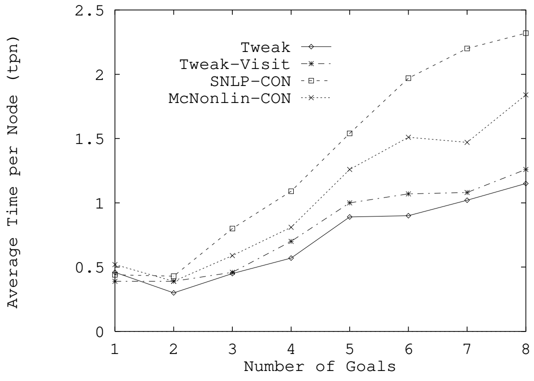
8.3.1 ART-MD-RD Experiments

In this experiment we used the problems from ART-MD-RD again. An interesting characteristic of ART-MD-RD domain is that the interactions between the actions in the domain are such that often there is only one ground operator sequence that is a solution for a given problem consisting of a set of goals $\{G_i\}$. Thus, as the number of goals increase, the search space size increases, but the number of solutions does not increase commensurately. This has the implicit effect of reducing solution density.

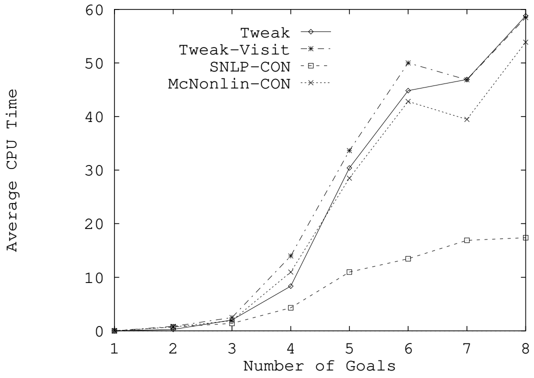
To compare the effect of protection strategies, we experimented with four planners, TWEAK, TWEAK-visit, SNLP-CON and McNONLIN-CON (described in



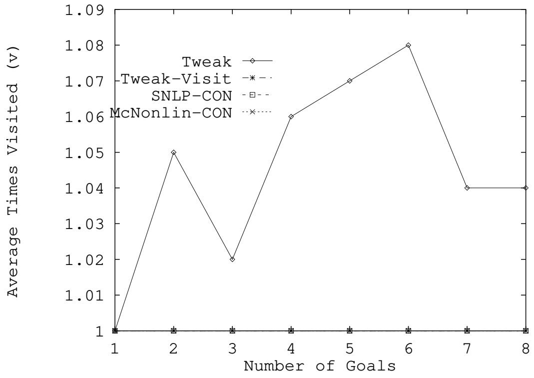
(a) Avg. # nodes searched $\mathcal{F}_d$ (LIFO)



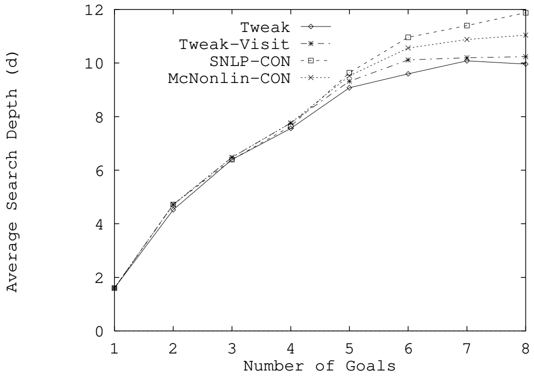
(b) Avg. T_{rp}



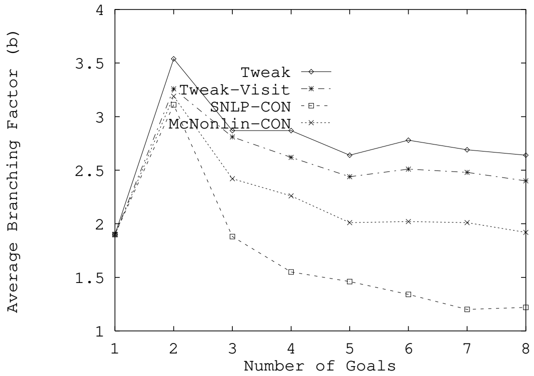
(c) Avg. CPU time (LIFO)



(d) Avg. v (LIFO)



(e) Avg. soln. depth (LIFO)



(f) Avg. branching factor (LIFO)

Fig. 16. Plots comparing relative effect of protection strategies ART-MD-RD experiments.

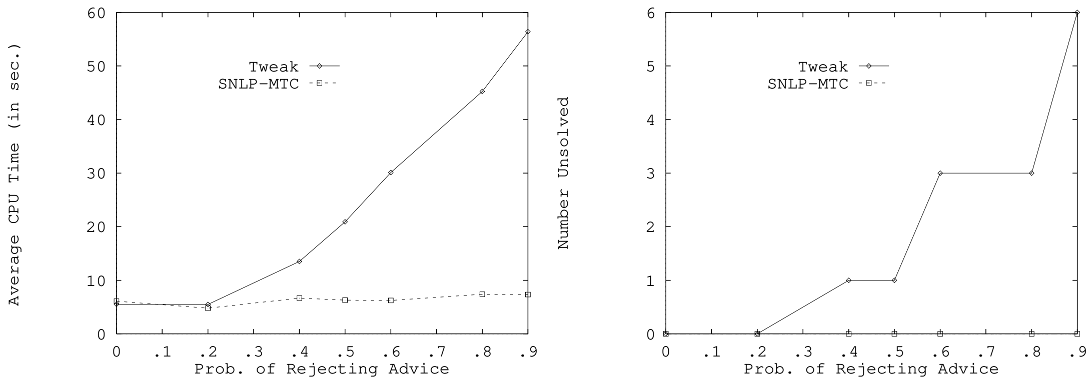
Section 5), all of which differ only along the dimension of bookkeeping constraints they employ.

We recall that TWEAK employs no protection strategy, TWEAK-visit uses agenda popping, McNONLIN-CON uses interval protection and SNLP-CON uses contributor protection. In terms of search space redundancy TWEAK allows both overlapping candidate sets, and repeated establishments of the same precondition. TWEAK-visit and McNONLIN-CON avoid repeated establishments, but allow overlapping candidate sets. The difference between TWEAK-visit and McNONLIN-CON is that the latter backtracks as soon as any establishment is necessarily violated, while the former will backtrack only when all the conditions returned by the goal-selection strategy have already been established once. SNLP-CON improves upon McNONLIN-CON by avoiding both repeated establishments and overlapping candidate sets. All these planners are *complete* for the problems in ART-MD-RD domain. [36]

As the problem size (i.e., number of goals) increases in ART-MD-RD, the solution density reduces and the search space size increases. From Hypothesis 4 in Section 7.2 we expect that for problems of small size, and thus high solution density, the performance of the different systems will be similar. Based on Hypothesis 5, we expect that as the problem size increases, the explored search space size of SNLP-CON will be the smallest, that of TWEAK will be the largest, and those of McNONLIN-CON and TWEAK-visit will fall in the middle.

Figure 16 shows the plots comparing the relative performance of these planners on problems in ART-MD-RD domain. We note that these plots conform to our predictions. In particular, as the problem size increase, the average search space size becomes significantly lower for planners with stronger protections strategies (Figure 16(a)). This reduction is well correlated with the average branching factors of the planners (Figure 16(f)). The plots of average CPU time in Figure 16(c) show that the the differences in it are not as drastic as the average search space sizes. This can be explained by the fact that the planners using stronger protection strategies impose more auxiliary constraints (IPCs in our case), and thus will take longer to check consistency of the partial plans. The plots of average per node refinement cost in Figure 16(b) confirm this explanation. Finally, the plots in Figure 16(d) confirm that of the four planners, only TWEAK can suffer from repeated establishment of the same precondition.

[36] Although, as discussed in Section 5.2, SNLP-CON, McNONLIN-CON and TWEAK-visit are not in general complete for MTC-based termination criteria, they *are* complete for problems in ART-MD-RD domain. This is because in ART-MD-RD, by the time the planners establish each precondition once, they would, of necessity, have a totally ordered partial plan, because of the causal ordering forced by the hf/he preconditions of the steps. Since eventually solution bearing partial plans will have only one ground linearization, and since all planners can eventually consider each precondition for establishment at least once, SNLP-Con, McNONLIN-CON and TWEAK-Visit will also eventually produce a partial plan all of whose ground linearizations correspond to solutions. Thus, they will be complete for MTC based termination condition



(a) Avg. CPU Time (b) Number Unsolved

Fig. 17. Plot showing the effect of misdirection on protection strategies

8.3.2 Misdirection Experiment

In this experiment, we compared TWEAK and SNLP-MTC in ART-MD, a variant of ART-MD-RD domain without the hf/he conditions (similar to $D^m S^1$ domain described in [2]). Both planners were started off with a *mingoals* heuristic [27], which ranks a partial plan by the number of preconditions that are not necessarily true according to MTC. We 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 Hypothesis 6, strong protection strategies should help in such situations.

The plot in Figure 17 shows the performance of the planners (measured in terms of average CPU time taken for solving a set of 20 random problems run on a SparcII with a time limit of 120 seconds), 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.

The experiments about protection strategies 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.

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 4
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

8.4 Combined effects of Protection strategies and Tractability refinements

In the last two sections, we looked at the individual effects of tractability refinements and protection strategies on performance. Given that most existing planners differ in *both* rather than only one of these dimensions, it would be interesting to understand which of these differences have the dominant effect on performance. The experiments in the previous sections provide a partial answer to this question.

Recall that the planners used in the experiments in Section 8.2.2 differ both in the tractability refinements they use and the protection strategies they employ (this is not surprising since conflict resolution refinements are related to the type of auxiliary constraints posted by the protection strategies). Table 4 shows the estimates of the average redundancy factors at the termination fringe of the five planners used in these experiments, for 30 6-goal ART-MD-RD problems.[37] As to be expected, SNLP-MTC and SNLP-UA, which use contributor protections, have no redundancy ($\rho_d = 1$). However, from the plots in Figure 13, 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.

Another data point can be obtained by comparing the relative performance of SNLP-CON and McNONLIN-CON to SNLP-MTC and McNONLIN-CON respectively. In particular, by comparing the plots of the size of explored search space size in Figure 16(a) and Figure 15(e), we note that the effect of tractability refinements dominates over that of protection strategies. While the average search space size of SNLP-CON is significantly lower than that of McNONLIN-CON (Figure 16(a)), the search-space sizes of SNLP-MTC and McNONLIN-MTC are almost identical (Figure 15(e)).

[37] The estimates were made by considering only the minimal candidates corresponding to the safe ground linearizations of the plans at the termination fringe.

From the foregoing, we note that in cases where two planners differ both in terms of tractability refinements and protection strategies, the empirical performance differentials are dominated more by the differences in the tractability refinements than 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. This latter observation is supported by the comparison between TWEAK and SNLP-MTC in the misdirection experiments reported in Section 8.3.2.

9 Related Work

We will partition the related work discussion loosely into 3 categories: relations with other refinement search-type frameworks, relations with other experimental analyses of planning algorithms and relations with other research efforts to improve/explain the performance of partial order planners.

9.1 Relations with other Refinement Search-based frameworks

In [36,37], Smith uses a search paradigm called global search to provide a unifying framework for scheduling. His motivation is to attempt to use the global search framework, in conjunction with an interactive software development system (KIDS [38]) to synthesize efficient scheduling algorithms for given problem populations [37]. Global search has several similarities with the refinement search framework discussed in this paper. For example, corresponding to our monotonic auxiliary constraints, Smith's global search has the notion of filters. In addition to the ideas of ''splitting'' and ''pruning'', Smith also talks about the notion of reducing the candidate set size through a process of ''constraint tightening'' or ''constraint propagation.'' In theory, such a process is also possible in the refinement search framework that is described here. We did not concentrate on this aspect as we are not aware of any existing planners that use constraint propagation in this fashion.

The idea of looking at partial plans not as incomplete solutions, but rather as sets of potential solutions, has also been used in Ginsberg's recent work on approximate planning [9]. In particular, his notion of ''linearizations'' of a plan is very close to our notion of candidate set of a partial plan. One technical difference is that Ginsberg makes no difference between candidates and solutions, assuming instead that a linearization that does not execute and produce the goals is an ''incorrect'' solution. He also espouses the view that the goal of planning is to produce a plan whose candidate set contains ''more'' solutions than it does non-solutions. Ginsberg also deals with the problem of non-finite candidates sets by introducing a new type of constraint called ''immediacy constraint''. In our framework, the immediacy constraint can be seen as a monotonic auxiliary constraint. Consider the constraint where the step t_1 is constrained to come immediately before t_2 (denoted by "$t_1 * t_2$"). A ground operator sequence S is said to satisfy this constraint with respect to a mapping $\mathcal{M}$ if $\mathcal{M}$ maps t_1 and t_2 to consecutive elements of S. It can be easily seen that a partial plan of the form $t_0 * t_1 * t_2 * t_\infty$ will have exactly one ground operator sequence that belongs to its candidate set.

9.2 Relations with other experimental analyses of planning algorithms

Let us now address the relations between our work, and previous work on comparative analysis of planning strategies. As we mentioned earlier, much of the previous work has concentrated on comparing partial order and total order planners. Two representative efforts of this line of research are those of Barrett and Weld [2] and Minton *et al.* [26,27].

Barrett and Weld [2] compare the performance of two plan-space (SNLP and TOCL, in Table 1), and a state-space planner (TOPI), and develop hypotheses regarding the relative performance of these planners. In particular, they extend the notions of serializability defined in [22] to include trivial and laborious serializability, and contend that the partial-order planners yield superior performance because the subgoals in planning domains are trivially serializable for them more frequently than for total order planners. Minton et al. [26,27] concentrate on the performance of a partial order and a total order plan-space planners (UA and TO in Table 1) and develop hypotheses regarding their relative performance.

From the point of view of `Refine-Plan`, it is interesting to note that both TOCL and SNLP, and UA and TO are pairs of plan-space planners that differ only in terms of the tractability refinements they employ. From Table 1, we can see that TOCL employs a more eager tractability refinement strategy compared to SNLP, while TO employs a more eager tractability refinement strategy compared to UA. This observation puts the empirical studies of Barrett and Weld and Minton et al. in perspective and provides a common ground for comparing their results with ours. In particular, our hypotheses regarding the effect of tractability refinements (Section 7.1) should be applicable for both these comparisons.

Another interesting relation with Minton et al.'s work concerns the comparison between TWEAK and UA. Minton et al. [26] suggest that UA could be more efficient than TWEAK because TWEAK has more redundancy in its search space. To begin with, the notion of redundancy used in their paper is very different from the notion of redundancy defined in terms of overlapping candidate sets, that we used in this paper. In particular, seen as instantiations of `Refine-Plan`, neither TWEAK nor UA uses any bookkeeping constraints (see Table 1), and thus suffer from both types of redundancy discussed in Section 4.3 (and Figure 8). They differ instead on the type of tractability refinements they use. Given this, and our hypotheses and experiments regarding the effect of tractability refinements on performance, there is no *a priori* reason to believe that UA will always outperform TWEAK; the relative performance will depend on the interaction between $b_t - b_e$ factors. In fact, the plots in Figure 13 show that TWEAK can out-perform UA in any domain where high-frequency conditions are not considered for establishment explicitly in the beginning parts of the search. The *unambiguity* of partial plans maintained by UA seems to have less of an effect on the performance. [38]

Another recent effort that did a comparative study on partial order planners is that of Yang and Murray [43]. They evaluate the utility of applying pruning heuristics to partial-order planners while preserving their completeness. One particular heuristic, known as *temporal coherence*, works by using a set of domain constraints to prune away plans that do not ''make sense,'' or temporally incoherent. their analysis shows that, while intuitively appealing, to maintain completeness, temporal coherence can only be applied to a very specific implementation of a partial-order planner. Furthermore, the heuristic does not always improve planning efficiency; in some cases, its application can actually degrade the efficiency of planning dramatically.

9.3 Relations with other efforts to explain/improve planning performance

Our unified framework for partial order planning also puts in perspective some of the recent efforts to improve the performance of partial order planners. An example is the recent work on improving planning performance through selective deferment of conflict resolution [32]. Since conflict resolution is an optional step in `Refine-Plan`, the planner can be selective about which conflicts to resolve, without affecting the completeness or the systematicity of `Refine-Plan`(see Section 4.5.3). [39] Conflict deferment is motivated by the idea that many of the conflicts are ephemeral, and will be resolved automatically during the course of planning. Thus, conflict deferment tends to reduce the search space size both by reducing the tractability branching factor b_t, and by pushing the branching from earlier parts of the search to the later parts of the search. However, this does not come without a tradeoff. Specifically, when the planner does such partial conflict resolution, the consistency check has to once again test for existence of safe ground

[38] Similarly, Minton et al. conjecture [27] that addition of causal links (i.e., book-keeping strategies) will increase the the cost of UA. Our empirical comparison of SNLP-UA and UA in ART-MD-RD domain (Section 8.2.2) provide a data point for checking this conjecture. They show that although the per-node cost of SNLP-UA is higher than UA (Figures 15(c) and 15(d)), the average search space size of SNLP-UA is lower than that of UA (Figures 15(e) and 15(f)), thus making their performances almost identical.

[39] Note, from Section 5.2.4 that SNLP-CON and McNONLIN-CON do not do any conflict resolution at all, and thus correspond to deferring all the conflicts indefinitely.

linearizations, rather than order and binding consistency (making consistency check intractable once again).

The planners with which Peot et al. [32] experimented all used ordering and binding consistency checks. Conflict deferment in such situations can lead to refinement of inconsistent plans, thereby reducing κ_d and increasing $|\mathcal{F}_d|$, and leading to loss of strong systematicity (Definition 7). In such cases, we would expect that best performance is achieved neither with eager conflict resolution, nor with full conflict deferment. This intuition is consistent with the empirical evidence provided by Peot et al. [32]. In particular, they found that deferring any conflict which has more than one way of resolution is better than deferring every conflict. In terms of `Refine-Plan`, the former strategy reduces the chance that the partial plan with deferred conflicts is inconsistent.

In our discussion of bookkeeping constraints, we observed that some of them, especially the interval protection and contributor protection strategies, incur a strong form of commitment to previous establishments. In [12,13], Kambhampati provides a generalization of single-contributor causal links, called multiple-contributor causal links, which attempt to reduce the commitment to specific establishments. As shown in Section 4.3, the multiple contributor causal links can be formalized in terms of disjunctions of interval preservation constraints. Kambhampati describes experiments comparing a variety of partial order planners using causal link based bookkeeping strategies, and shows that multi-contributor causal links can avoid the harmful effects of the increased commitment of the single-contributor causal links to some extent.

The unifying framework also has clear pedagogical advantages in terms of clarifying the relations between many brand-name planning algorithms, and eliminating several long-standing misconceptions. An important contribution of `Refine-Plan` is the careful distinction it makes between bookkeeping constraints or protection strategies (which aim to reduce redundancy), and tractability refinements (which aim to shift complexity from refinement cost to search space size). This distinction removes many misunderstandings about plan-space planning algorithms. For example, it clarifies that the only motivation for total ordering plan-space planners is tractability of refinement. Similarly, it has been previously believed (e.g. [19]) that the systematicity of SNLP *increases* the average depth of the solution. Viewing SNLP as an instantiation of `Refine-Plan` template, we see that it corresponds to several relatively independent instantiation decisions, only *one* of which, viz., the use of contributor protections in the bookkeeping step, has a direct bearing on the systematicity of the algorithm. From the discussion in Section 4, it should be clear that the use of contributor protection does not, *ipso facto*, increase the solution depth in any way. Rather, the increase in solution depth is an artifact of the particular solution constructor function, and the conflict resolution and/or the preordering strategies used in order to get by with tractable termination and consistency checks. These can be replaced without affecting the systematicity property.

Similarly, our framework not only clarifies the relation between the unambiguous planners such as UA [27] and causal-link based planners such as SNLP [24], it also suggests fruitful ways of integrating the ideas in the two planning techniques (cf. SNLP-UA in Section 5.2.2).

10 Conclusion

The primary contribution of this paper is a unified framework for understanding and analyzing the design tradeoffs in partial-order plan-space planning. We started by providing a refinement search-based semantics to partial order planning, and developing a generic for refinement planning algorithm. We then showed that most existing plan-space planners are instantiations of this algorithm. Next, we 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. Based on this understanding, we have developed hypotheses regarding the effect of two of these choices -- tractability refinements and protection strategies -- on the relative performance of different planners. Finally, we described several focused empirical studies to both evaluate our hypotheses and demonstrate the explanatory power of our unified framework. These studies show that the performance is affected more by the differences in tractability refinements than by the differences in protection strategies.

Limitations and Future Work: 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. Specifically, we need to develop hypotheses regarding the effects of other components of the `Refine-Plan` algorithm on performance. We also need to develop a more complete understanding about the second-order interactions between the various parts of the `Refine-Plan` algorithm and their effect on the performance. Much of the analysis in this paper has concentrated on breadth-first search regimes. Given that depth-first backtracking search regimes are more practical in terms of their space requirements, we need to extend our analysis to cover these cases. A preliminary analysis of the effect of tractability refinements on the success probability of `Refine-Plan` in depth-first search regimes is given in [11,15]. This analysis shows that if we assume that tractability refinements split the candidate set of the partial plan in such a way that the solution candidates in it are randomly distributed among the resulting refinements, then we would expect the probability of success in any given search branch to reduce with more eager tractability refinements. This analysis needs to be validated, as well as extended to cover situations where refinements do not distribute solution candidates randomly among the refined plans.

In Section 9, we noted that Smith [37] uses a global search based framework to automatically synthesize customized schedulers. Given the close similarity between the global search and refinement search frameworks, this raises the intriguing possibility: *is it possible to automatically synthesize the instantiation of* `Refine-Plan` *that will be most efficient in solving a given class of planning problems?* In collaboration with Smith, we are currently evaluating the feasibility of this approach.

Finally, many existing industrial strength planners use task-reduction techniques, which are not modeled by our `Refine-Plan` algorithm. In [17] we show that `Refine-Plan` can be extended to cover task reduction planners (also called HTN planners) by simply replacing the establishment refinement with another called task-reduction refinement (we also show that this model clarifies several features of HTN planning). Since the tractability refinements and protection strategies remain unchanged in the new algorithm, the insights regarding the their effect on performance, gained in this research, will also be applicable in the context of HTN planning. The extended algorithm could also serve as a basis for understanding other performance tradeoffs among task-reduction planners.

Acknowledgements

This paper is the result of combining and extending two initially independent research efforts, one by Kambhampati [11,15--17] and the other by Knoblock and Yang [19,21,20]. We thank the editors of this special issue for bringing us together. The paper by Knoblock and Yang [21] won the CSCSI (Canadian Society for Computational Studies of Intelligence) Best Paper Award in 1994 (sponsored by the Artificial Intelligence Journal).

Thanks are due to David McAllester for many enlightening (e-mail) discussions on the nature of refinement search [25]; and Bulusu Gopi Kumar, Suresh Katukam, Laurie Ihrig, Mark Drummond, Kutluhan Erol, Nort Fowler, Jonathan Gratch, Dan Weld, and the anonymous reviewers for critical comments on earlier versions of this paper.

Kambhampati's research is supported in part by an NSF Research Initiation Award IRI-9210997, an NSF Young Investigator Award IRI-9457634, and ARPA/Rome Laboratory planning initiative under grant F30602-93-C-0039. Knoblock's research is supported by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under contract no. F30602-91-C-0081. Yang's research is supported in part by grants from the Natural Science and Engineering Research Council of Canada, and ITRC: Information Technology Research Centre of Ontario.

A List of Symbols

Symbol	Denotes
$\mathcal{A}$	Agenda (a list of the various preconditions the planner intends to establish)
$\langle c@s \rangle$	a point truth constraint (PTC)
d	solution depth
b	branching factor
l	length
`fin`	The dummy operator corresponding to the goal state ($\mathcal{ST}$ maps t_∞ to `fin`)
g, G	top level goals of the plan (preconditions of t_∞).
$\mathcal{L}$	auxiliary constraints in the partial plan
$\mathcal{L}_\emptyset$	auxiliary constraints in the null plan $\mathcal{P}_\emptyset$ (contains a PTC $\langle g@t_\infty \rangle$ for each top-level goal g of the plan).
c, p, q, r	denote preconditions and effects of steps in the plan
o, o_i	operators in the domain
$\mathcal{S}_\mathcal{G}$	The solution criterion for a refinement search
T	set of all steps in a partial plan
$\mathcal{N}$	Nodes in refinement search
$\mathcal{N}_\emptyset$	The node whose candidate set corresponds to candidate space
$\mathcal{P}$	a partial plan
$\mathcal{P}_\emptyset$	a null partial plan (with which `Refine-Plan` starts). $\langle\langle \mathcal{P}_\emptyset \rangle\rangle = \mathcal{K}$.
$\langle\langle . \rangle\rangle$	Candidate set (of a node or a partial plan)
$\mathcal{R}$	A refinement operator
R	The set of refinement operators of a refinement search
$\langle s, c, s' \rangle$	an interval preservation constraint (IPC)
$\mathcal{M}$	a mapping between steps of a plan and the elements of a ground operator sequence
$\mathcal{ST}$	a mapping between the steps of a plan, and the operators they stand for
S	a ground operator sequence
$\rightarrow$	''maps''
`start`	The dummy operator corresponding to the initial state ($\mathcal{ST}$ maps t_0 to `start`)
t, s	steps in a partial plan
t_∞	the special step that follows all other steps of a partial plan
t_0	The special step that precedes all other steps of a partial plan
$\mathcal{K}$	The candidate space
$\lvert . \rvert$	Cardinality of a set
$\emptyset$	The empty set

Additional list of symbols used in the complexity analysis can be found in Table 3.

B Glossary of Terms

This section gives a quick and informal glossary of terms. For more detailed definitions, look at the appropriate sections of the text where the terms are defined.

Agenda A datastructure used by `Refine-Plan` algorithm to keep track of the list of preconditions $\langle c, s\rangle$ of the plan that need to be handled through establishment refinement.

Agenda Popping A bookkeeping strategy where `Refine-Plan` removes from agenda each precondition $\langle c, s\rangle$ as soon as it has been considered by establishment refinement once. When the agenda is empty, the corresponding partial plan is pruned. This strategy ensures that no precondition is considered for establishment more than once.

Auxiliary Constraints Constraints of a partial plan other than steps, orderings and bindings. The auxiliary constraints can be divided into two broad classes -- monotonic and non-monotonic. `Refine-Plan` handles two specific types of auxiliary constraints -- *interval preservation constraint* (IPC), which is monotonic; and *point truth constraint* (PTC), which is non-monotonic.

Bookkeeping Strategy The strategy used by a planner to keep track of the goals it already established (also called *protection strategy*). See Section 4.3.

Candidate A potential solution for a problem. For planning, a ''ground operator sequence.''

Candidate of a Partial Plan A ground operator sequence that contains all the steps of the plan in an ordering and binding consistent with the partial plan, and which satisfies the monotonic auxiliary constraints of the partial plan.

Candidate Space The space of potential solutions for a problem.

Causal Link A concept due to McAllester [24]. In terms of the terminology of this paper, a causal link $s \xrightarrow{p} s'$ can be seen as a *macro*-constraint that corresponds to the following constraints: (i) $s \prec s'$ (ii) s gives an effect p (this may involve either addition of binding constraints, or secondary preconditions, in the form of PTCs to s) (iii) an IPC $\langle s, p, s'\rangle$ to ensure that p is preserved between s and s'.

Conflict An IPC $\langle t, p, t'\rangle$ and a step (called threat) t'' of the plan such that t'' can possibly come between t and t', and t'' deletes p.

Conflict Resolution Refinements A type of tractability refinements that order steps in the partial plan based on how they interact with the IPCs of the plan. If the partial plan contains an IPC $\langle s, p, s'\rangle$ and the step s'' has an effect $\neg p$, then conflict resolution will attempt to order s'' to either come before s or after s'.

Ground Linearization An operator sequence that matches a topological sort of the steps of the partial plan, with all variables instantiated.

Interval Preservation Constraint (IPC) An monotonic auxiliary constraint on a partial plan. An IPC $\langle s, p, s''\rangle$ of a partial plan $\mathcal{P}$ requires that every candidate of the partial plan must preserve p between the operators corresponding to s and s'.

Partial Plan A constraint set that specifies a sets of steps, orderings among those steps, bindings among the variables, a mapping between the steps and the domain operators, and a set of auxiliary constraints.

Point Truth Constraint (PTC) A non-monotonic auxiliary constraint on a partial plan. A PTC $\langle c@s \rangle$ of a partial plan $\mathcal{P}$ requires that every solution of the partial plan $\mathcal{P}$ must have c true before the operator corresponding to s.

Preconditions of a Plan Tuples $\langle c, s \rangle$, where s is a step in the plan, and c is a condition that needs to be true before s (c may either be a precondition of the operator corresponding to s or a secondary precondition added during establishment).

Preordering Refinements A type of tractability refinements that order steps in the partial plan based only on the static description of their effects and preconditions (and without taking into account the specific roles they play in the current partial plan). Two important pre-ordering strategies are ''total ordering'', which orders every pair of steps in the plan, and ''unambiguous ordering'' which orders every pair of steps that have overlapping preconditions and effects (signifying their *potential* to interact with each other).

Protection Strategy See *bookkeeping strategy*

Refinement Operator An operator that splits the candidate set of a search node (in the case of planning, the candidate set of a partial plan). `Refine-Plan` uses two types of refinement operators: the establishment refinements, and the tractability refinements.

Safe Ground Linearization A ground linearization that satisfies the monotonic auxiliary constraints.

Search Node A constraint set that implicitly stands for the set of candidates that satisfy the constraints. In the case of planning, a partial plan is a search node.

Secondary Preconditions Preconditions that need to be made true before an action so as to make the action provide a particular effect. For example, if an action a has a conditional effect `If` p `then` q, then we can make a give the effect q by making p a secondary precondition of a. Similarly, we can make a give $\neg a$ by making $\neg p$ a secondary precondition of a. In `Refine-Plan` framework, such secondary preconditions become PTCs for the action (e.g. $\langle p@a \rangle$ or $\langle \neg p@a \rangle$).

Solution A candidate that solves the problem. For planning, a ground operator sequence that can be executed from the initial state, and gives rise to a state where all the goals of the plan are satisfied.

Solution Constructor A procedure which takes a search node and the goals of the problem, and checks to see if one of the candidates of the search node is a solution for the problem. The procedure may (i) return a solution or (ii) signal that there are no solutions, or (iii) signal that it is unable to decide either way. In the first case, search is terminated with success. In the second case, the search node is pruned, and in the third case the search node is refined further.

Solution of a Partial Plan A candidate of a partial plan $\mathcal{P}$ that also solves the planning problem. This requires that the candidate be (i) executable from the initial state (given by the effects of t_0) (ii) satisfy all the non-monotonic auxiliary constraints.

Solution Node/Solution Plan A search node (partial plan) for which the solution constructor returns a solution, leading to the termination of search.

Strong Systematicity The property that the search is systematic and no node with an empty candidate set has children in the search tree. In a refinement search that is strongly systematic, the size of the fringe is bounded by the size of the candidate space.

Systematicity The property that the candidate sets of search nodes in different branches of the search tree are non-overlapping.

Tractability Refinement The class of refinements whose primary purpose is to reduce the cost of consistency check. They do this by either reducing the number of ground linearizations per partial plan, or by making all ground linearizations have identical properties with respect to the auxiliary constraints. Two important types of tractability refinements are *preordering* refinements and *conflict resolution* refinements.

References

[1] J.A. Ambros-Ingerson and S. Steel. Integrating Planning, Execution and Monitoring. In *Proc. 7th AAAI*, 1988.

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

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

[4] G. Collins and L. Pryor. Achieving the functionality of filter conditions in partial order planner. In *Proc. 10th AAAI*, 1992.

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

[6] K. Erol, D. Nau and J. Hendler. Toward a general framework for hierarchical task-network planning. In *Proc. of AAAI Spring Symp. on Foundations of Automatic Planning*. 1993.

[7] O Etzioni and R Etzioni. Statistical Methods for Analyzing Speedup Learning Experiments. Machine Learning, 14(3), 1994.

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

[9] M. Ginsberg. Approximate Planning. *Artificial Intelligence*, Special Issue on Planning, Scheduling and Control. 1995.

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

[11] S. Kambhampati. Planning as Refinement Search: A unified framework for comparative analysis of Search Space Size and Performance. Technical Report 93-004, Arizona State University, June, 1993. Available via anonymous ftp from `enws318.eas.asu.edu:pub/rao/papers.html`

[12] S. Kambhampati. Multi-Contributor Causal Structures for Planning: A Formalization and Evaluation. *Artificial Intelligence*, Vol. 69, 1994. pp. 235-278.

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

[14] S. Kambhampati and D.S. Nau. On the Nature and Role of Modal Truth Criteria in Planning. *Artificial Intelligence*, 1994 (to appear). (Available as Tech. Report. ISR-TR-93-30. University of Maryland, March, 1993. A shorter version appears in *Proc. 12th Natl. Conf. on AI (AAAI-94)*, July 1994)

[15] 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.

[16] S. Kambhampati. Design Tradeoffs in Partial Order (Plan Space) Planning. In *Proc. 2nd Intl. Conf. on AI Planning Systems (AIPS-94)*, June 1994.

[17] S. Kambhampati. A comparative analysis of partial order planning and task reduction planning SIGART Bulletin, Special Section on evaluating plans, planners and planning agents. Vol. 6, No. 1, January 1995.

[18] C. Knoblock. Automatically Generating Abstractions for Planning. *Artificial Intelligence*, Vol. 68, 1994.

[19] 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*. 1993.

[20] C. Knoblock and Q. Yang. Relating the Performance of Partial-order planning algorithms to domain features. SIGART Bulletin, Special Section on evaluating plans, planners and planning agents. Vol. 6, No. 1, January 1995.

[21] C. Knoblock and Q. Yang. Evaluating the Tradeoffs in Partial-Order Planning Algorithms. In *Proc. Canadian Conference on Artificial Intelligence (AI-94)*, May, 1994.

[22] R. Korf. Planning as Search: a quantitative approach. *Artificial Intelligence*, Vol. 33, 1987. pp. 65-88.

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

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

[25] D. McAllester. Private Communication. Spring 1993.

[26] S. Minton, J. Bresina and M. Drummond. Commitment Strategies in Planning: A Comparative Analysis. In *Proc. 12th IJCAI*, 1991.

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

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

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

[30] E.P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the Situation Calculus. Proc. 1st Intl. Conf. on the Ppls. of Knowledge Rep. and Reasoning. 1989.

[31] E.P.D. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context dependent effects. In *Proc. IJCAI-91.* , 1991.

[32] M.A. Peot and D.E. Smith. Threat-Removal Strategies for Nonlinear Planning. In *Proc. Eleventh AAAI*, 1993.

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

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

[35] D.E. Smith and M.A. Peot. Postponing threats in partial-order planning. In *Proc. Eleventh AAAI*, 1993.

[36] D.R. Smith. Structure and Design of global search algorithms. Tech. Rep. KES.U.87.13, Kestrel Institute, November 1987. To appear in *Acta Informatica*.

[37] D.R. Smith and E.A. Parra. Transformational approach to transportation scheduling. In *Proc. 8th Knowledge-based software engineering Conference*, 1993.

[38] D.R. Smith KIDS -- a semi-automatic program development system. *IEEE Trans. on Software Engineering*, Vol. 16, No. 9, 1990. pp. 1024-1043.

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

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

[41] Q. Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence Journal*, 6:12--24, 1990.

[42] Q. Yang, J. Tenenberg, and S. Woods. Abstraction in nonlinear planning. Technical Report CS 91-65, University of Waterloo, Department of Computer Science, Waterloo, Ontario, Canada N2L 3G1, 1991.

[43] Q. Yang and C. Murray. An evaluation of the temporal coherence heuristic in partial-order planning. *Computational Intelligence Journal*, 10(3), 1994.