To appear in Proc. Principles of Knowledge Rep. and Reasoning (KRR), May 1994

Refinement Search as a Unifying Framework for analyzing Planning Algorithms

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

Abstract

Despite the long history of classical planning, there has been very little comparative analysis of the performance tradeoffs offered by the multitude of existing planning algorithms. This is partly due to the many different vocabularies within which planning algorithms are usually expressed. In this paper, I show that refinement search provides a unifying framework within which various planning algorithms can be cast and compared. I will provide refinement search semantics for planning, develop a generalized algorithm for refinement planning, and show that all planners that search in the space of plans are special cases of this algorithm. I will then show that besides its considerable pedagogical merits, the generalized algorithm also (*i*) allows us to develop a model for analyzing the search space size, and refinement cost tradeoffs in plan space planning, (*ii*) facilitates theoretical and empirical analyses of competing planning algorithms and (*iii*) helps in synthesizing new planning algorithms with more favorable performance tradeoffs. I will end by discussing how the framework can be extended to cover other planning models (e.g. state-space, hierarchical), and richer behavioral constraints.

1 Introduction

The idea of generating plans by searching in the space of (partially ordered or totally ordered) plans has been around for almost twenty years, and has received a lot of formalization in the past few years. Much of this formalization has however been limited to providing semantics for plans and actions, and proving soundness and completeness 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.[1] 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.

In this paper, I shall show that viewing planning as a refinement search provides a unified framework within which the complete gamut of plan-space planning algorithms can be effectively cast and compared.[2] I will start by characterizing planning as a refinement search, and provide semantics for partial plans and plan refinement operations. I will then provide a generalized algorithm for refinement planning, in terms of which the whole gamut of the so-called plan-space planners can be expressed. The different ways of instantiating this algorithm correspond to the different design choices for plan-space planning. This unified view facilitates separation of important ideas underlying individual algorithms from ''brand-names'', and thus provides a rational basis for understanding the design tradeoffs and fruitfully integrating the various approaches. I will demonstrate this by using the framework as a basis for analyzing search space size vs. refinement cost trade-offs in plan-space planning, and developing novel planning algorithms with interesting performance tradeoffs.

The paper is organized as follows: Section 2 provides the preliminaries of refinement search, develops a model for

*This research is supported in part by an NSF Research Initiation Award IRI-9210997, and ARPA/Rome Laboratory planning initiative under grant F30602-93-C-0039. Special thanks to David McAllester for many enlightening (e-mail) discussions on refinement search, and Bulusu Gopi Kumar for critical comments.

[1] The work of Barrett and Weld [1] as well as Minton et. al. [16, 17] are certainly steps in the right direction. However, they do not tell the full story since the comparison there was between a specific partial order and total order planner. The comparison between different partial order planners itself is still largely unexplored.

[2] 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.

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 candidate set of a partial plan is formally defined in this section, and the ontology of constraints used in representing partial plans is described. Section 4 describes the generalized refinement planning algorithm, discusses its various components, and explains how the existing plan-space planners can all be seen as instantiations of the generalized algorithm. Section 5 discusses the diverse applications of the unifying componential view provided by the generalized algorithm. Specifically, we will see how the unifying view helps in explicating and analyzing the tradeoffs (Section 5.1), facilitating comparative performance analyses (Section 5.2), and synthesizing planning techniques with novel performance tradeoffs (Section 5.3). Section 6 discusses how the generalized algorithm can be extended to handle richer types of goals (e.g. maintenance goals, intermediate goals), and other types of planning models (e.g. HTN planning, state-space planning). Section 7 presents the concluding remarks.

2 Refinement search Preliminaries

A refinement search (or split-and-prune search [18]) can be visualized as a process of starting with the set of *all* potential candidates for solving the problem, and splitting the set repeatedly until a solution candidate 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 potential candidates, denoted by $\langle\langle \mathcal{N} \rangle\rangle$.

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 potential candidates (we shall call this set $\mathcal{K}$).

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 -- they map a search node $\mathcal{N}$ to a set of children nodes $\{\mathcal{N}'_i\}$ such that $\forall i \ \langle\langle \mathcal{N}'_i \rangle\rangle \subseteq \langle\langle \mathcal{N} \rangle\rangle$.

Definition 1 *Let* **R** *be a refinement strategy that maps a node $\mathcal{N}$ to a set of children nodes $\{\mathcal{N}'_i\}$.* **R** *is said to be* **complete** *if $\bigcup_i \langle\langle \mathcal{N}'_i \rangle\rangle = \langle\langle \mathcal{N} \rangle\rangle$ (i.e., no candidate is lost in the process of refinement).*

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

The search terminates when a node $\mathcal{N}$ is found for which the solution constructor returns a solution candidate. 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:

1. *fail*, meaning that no candidate in $\langle\langle \mathcal{N} \rangle\rangle$ satisfies the solution criterion
2. Some candidate $c \in \langle\langle \mathcal{N} \rangle\rangle$ which satisfies the solution criterion (i.e., c is a solution candidate)
3. $\perp$, meaning that `sol` can neither construct a solution candidate, nor determine that no such candidate exists.

Algorithm: Refinement Search(`sol`, R)
Initialize `open` with $\mathcal{N}_\emptyset$, the node with initial (null) constraint set
Begin Loop
If `open` is empty, terminate with failure
Else, non-deterministically pick a node $\mathcal{N}$ from `open`
If `sol`$(\mathcal{N}, \mathcal{G})$ returns a candidate c,
Then return it with `success`
Else, choose some refinement operator $\mathcal{R} \in$ **R**,
(*Not a backtrack point.*)
Generate $\mathcal{R}(\mathcal{N})$, the refinements of n with respect to $\mathcal{R}$.
Prune any nodes in $\mathcal{R}(\mathcal{N})$ that are inconsistent.
Add the unpruned nodes in $\mathcal{R}(\mathcal{N})$ to `open`.
End Loop

Figure 1: An algorithm for generic refinement search

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 `sol`$(\mathcal{N}, \mathcal{S}_\mathcal{G})$ returns a solution candidate.

Definition 2 (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 candidate c of the problem, there exists some search node $\mathcal{N}$ that results from a finite number of successive refinement operations on $\mathcal{N}_\emptyset$ (the initial search node whose candidate set is the entire candidate space), such that* `sol` *can pick up c from $\mathcal{N}$.*

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. [6]) such that every potential 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. Anytime the set of constraints of a search node becomes inconsistent (unsatisfiable), the candidate set becomes empty, and the node can be pruned.

Definition 3 (Inconsistent Search Nodes) *A search node is said to be inconsistent if its candidate set is empty, or equivalently, its constraint set is unsatisfiable.*

Search Space Size: Figure 1 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. 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 b^d. Thus, we have,

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

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 4 (Systematicity and Strong 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). Additionally, the search is said to be* **strongly systematic** *if it is systematic and never refines an inconsistent node.*

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 potential candidates $\mathcal{K}$. For strongly systematic search, in addition to ρ being equal to 1, we also have $\kappa \geq 1$ (since no node has an empty candidate set) and thus $|\mathcal{F}_d| \leq |\mathcal{K}|$. Thus,

Proposition 1 *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 and only 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

In this section, we shall develop a formal account of plan-space planning as a refinement search. Whatever the exact nature of the planner, the ultimate aim of (classical) planning is to find a *ground operator sequence*, 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 sub-class of behavioral constraints called the goals of attainment [5], which essentially constrain 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 (the exception is Section 6, which shows that our framework can be easily extended to a richer class of goals).

The operators (aka actions) in classical planning are modeled as general state transformation functions. We will be assuming that the domain operators are described in ADL [19, 20] representation with *Precondition* and *Effect* formulas. The precondition and effect formulas are *function-less* 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. [2, 10]).[3]

From the above definitions, it is clear that any potential solution for a planning problem must be a ground operator sequence. Thus, viewed as a refinement search, the candidate space, $\mathcal{K}$, of a planning problem, is the set of all ground operator sequences. As an example, if the domain contains three ground actions $a1$, $a2$ and $a3$, then the regular expression $\{a1|a2|a3\}^*$ would describe the candidate space for this domain. We will next define when a ground operator sequence is considered a solution to a planning problem in classical planning:

Definition 5 (Plan Solutions) *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:*

1. *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*
2. *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.*

Traditionally, the *completeness* of a planner is measured in terms of its ability to find minimal solutions (cf. [22, 19, 14]):

Definition 6 (Planner Completeness) *A planning algorithm is said to be* **complete** *if it can find all minimal solutions for every solvable problem.*

3.1 Refinement Search Semantics for Partial Plans

When plan-space planning is viewed as a refinement search, the constraint sets associated with search nodes can be seen as defining partial plans (in the following, we will be using the terms ''search node'' and ''partial plan'' interchangeably). The candidate set of a partial plan will be defined as all the ground operator sequences that satisfy the partial plan constraints.

The partial plan representation used by refinement planners can be described in terms of a 6-tuple: $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}, \mathcal{A}\rangle$ where:

- T is the set of steps in the plan; T contains two distinguished steps t_0 and t_∞.
- $\mathcal{ST}$ is a symbol table, which maps steps to domain operators. The special step t_0 is always mapped to the dummy operator `start`, and similarly t_∞ is always mapped to `fin`. The effects of `start` and the

[3] 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.

preconditions of `fin` correspond, respectively, to the initial state and the desired goals (of attainment) of the planning problem.

- O is a partial ordering relation over T.
- $\mathcal{B}$ is a set of codesignation (binding) and non-codesignation (prohibited bindings) constraints on the variables appearing in the preconditions and post-conditions of the operators.
- $\mathcal{L}$ is a set of auxiliary constraints. Auxiliary constraints are best seen as putting restrictions on the ground operator sequences being represented by the partial plan (see below).
- $\mathcal{A}$ is the set of preconditions of the plan, which are tuples of the form $\langle c, s \rangle$, where c is a condition that needs to be *made* true before the step $s \in T$. These include the preconditions and secondary preconditions [19] of all the actions introduced during planning process (see Section 4). $\mathcal{A}$ is sometimes referred to as the agenda of the plan.

Informally, the candidate set of a partial plan, $\mathcal{P}$, is the set of all ground operator (action) sequences that are consistent with the step, ordering, binding and auxiliary constraints of $\mathcal{P}$. Before we can formalize this notion, we need a better characterization of auxiliary constraints.

Auxiliary Constraints: Informally, auxiliary constraints should be seen as the constraints that need to be true for a ground operator sequence to belong to the candidate set of a partial plan. They can all be formalized as unary predicates on ground operator sequences. We will distinguish two types of auxiliary constraints: *monotonic* constraints and *non-monotonic* constraints.

Definition 7 (Monotonic Auxiliary Constraints) *An auxiliary constraint $\mathcal{C}$ is monotonic if given a ground operator sequence S that does not satisfy $\mathcal{C}$, no operator sequence S' obtained by adding additional ground operators to S will satisfy $\mathcal{C}$.*

Monotonic constraints are useful because of the pruning power they provide. If none of the ground operator sequences matching the ground linearizations of a partial plan satisfy its monotonic constraints, then that partial plan cannot have a non-empty candidate set, and thus can be pruned. For this reason, we will call the set of monotonic auxiliary constraints of a partial plan its **auxiliary candidate constraints** ($\mathcal{L}_c$), and the set of non-monotonic auxiliary constraints of a partial plan are called **auxiliary solution constraints**, ($\mathcal{L}_s$). Although auxiliary solution constraints cannot be used to prune partial plans, they can be used as a basis for selection heuristics during search (see the discussion of MTC-based goal selectors in Section 4.2, and that of filter conditions in 6).

Almost all of the auxiliary constraints employed in classical planning can be formalized in terms of two primitive types of constraints: *interval preservation constraints* (IPCs), and *point truth constraints* (PTCs):

Definition 8 (Interval Preservation Constraint) *An interval preservation constraint, $\langle s_i, c, s_j \rangle$ of a plan $\mathcal{P}$ is said to be satisfied by a ground operator sequence S according to a mapping function $\mathcal{M}$ that maps steps of $\mathcal{P}$ to elements of S, if and only if every operator o in S that comes between $\mathcal{M}(s_i)$ and $\mathcal{M}(s_j)$ preserves the condition c (i.e., if c is true in the state before o, then c will be true in the state after its execution).*[4]

Definition 9 (Point Truth Constraint) *A point truth constraint $\langle c@s \rangle$ is said to be satisfied by a ground operator sequence S with respect to a mapping $\mathcal{M}$ that maps steps of $\mathcal{P}$ to elements of S, if and only if either c is true in the initial state, and is preserved by every action of S occurring before $\mathcal{M}(s)$, or c is made true by some action $S[j]$ that occurs before $\mathcal{M}(s)$, and is preserved by all the actions between $S[j]$ and $\mathcal{M}(s)$.*

It is easy to see that interval preservation constraints are monotonic constraints, while point truth constraints are non-monotonic. In our model of refinement planning, IPCs are used to represent book-keeping (protection) constraints (Section 4.3) while PTCs are used to represent the solution constraints. In particular, given any partial plan $\mathcal{P}$, *corresponding to every precondition $\langle C, s \rangle$ on its agenda, the partial plan contains an auxiliary solution constraint $\langle C@s \rangle$.*[5]

We are now ready to formally define the candidate set of a partial plan:[6]

Definition 10 (Candidate set of a Partial plan) *Given a partial plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}, \mathcal{A} \rangle$, a ground operator sequence S is said to belong to $\mathcal{P}$'s candidate set, $\langle\langle \mathcal{P} \rangle\rangle$, if and only if there exists a mapping function $\mathcal{M}$ (called candidate mapping) that maps steps of $\mathcal{P}$ (excepting the dummy steps t_0 and t_∞) to elements of S, such that S satisfies all the constraints of $\mathcal{P}$ under the mapping $\mathcal{M}$. That is,*

1. *$\mathcal{M}$ is consistent with $\mathcal{ST}$. That is, if $\mathcal{M}$ maps the step s to $S[i]$ (i.e., the i^{th} element in the operator sequence), then $S[i]$ corresponds to the same action as $\mathcal{ST}(s)$.*
2. *$\mathcal{M}$ is consistent with the ordering constraints O and the binding constraints $\mathcal{B}$. For example, if $s_i \prec s_j$, and $\mathcal{M}(s_i) = S[l]$ and $\mathcal{M}(s_j) = S[m]$, then $l < m$.*
3. *S satisfies all the auxiliary candidate constraints ($\mathcal{L}_c$) under the mapping $\mathcal{M}$.*

Definition 11 (Solution Candidate of a Partial Plan) *A ground operator sequence S is said to be a solution candidate of a partial plan $\mathcal{P}$, if S is a candidate of $\mathcal{P}$ and S satisfies all the auxiliary solution constraints of $\mathcal{P}$.*

[4] Note that the plan does not have to *make* c true.

[5] Notice that this definition separates that the agenda preconditions from solution constraints. Under this model, the planner can terminate without having explicitly worked on the preconditions in the agenda (as long as the solution constraints are all true). Similarly, it also allows us to post solution constraints that we do not want the planner to explicitly work on (see the discussion about *filter conditions* in Section 6).

[6] Note that by our definition, a candidate of a partial plan may not be executable. It is possible to define candidate sets only in terms of executable operator sequences (or ground behaviors). We will stick with this more general notion of candidates, since coming up with an executable operator sequence can it self be seen as part of planning activity.

Given the definitions above, and the assumption that corresponding to every precondition of the plan, there exists a point truth constraint on the auxiliary solution constraints, we can easily prove the following relation between solution candidates and solutions of a planning problem:

Proposition 2 *Let $\mathcal{I}$ be the effect formula of t_0, and $\mathcal{G}$ be the precondition formula of t_∞ of $\mathcal{P}$. If a ground operator sequence S is a solution candidate of a partial plan $\mathcal{P}$, then S solves the problem $[\mathcal{I}, \mathcal{G}]$ according to Definition 5.*

Example: To illustrate the definitions above, suppose the partial plan $\mathcal{P}$ is given by the constraint set below, where the auxiliary constraints are interval preservation constraints as described above (the agenda field is omitted for simplicity):

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

Consider the ground operator sequence $S : o_1 o_3 o_2$. It is easy to see that as long as the action o_3 preserves p, S will belong to the candidate set of $\mathcal{P}$. This is because there exists a candidate mapping, $\mathcal{M} : \{t_1 \rightarrow S[1], t_2 \rightarrow S[3]\}$ according to which S satisfies all the constraints of $\mathcal{P}$ (the interval preservation constraint $\langle t_1, p, t_2\rangle$ is satisfied as long as o_3 preserves p). Similarly, the ground operator sequence $S' : o_1 o_2 o_5$ belongs to the candidate set of $\mathcal{P}$ if and only if o_5 preserves q.

Search Space Size: Search space size of a refinement planner can be estimated with the help of Eqn. 1. A minor problem in adapting this equation to planning is that according to the definitions above, both candidate space and candidate sets can have infinite cardinalities even for finite domains. However, if we restrict our attention to minimal solutions, then it is possible to construct finite versions of both. Given a planning problem instance P, let l_m be the length of the longest ground operator sequence that is a minimal solution of P. Let $\mathcal{K}$ be the set of all ground operator sequences of up to length l_m. $|\mathcal{K}|$ provides an upper bound on the number of operator sequences that need to be examined to ensure that all minimal solutions for the planning problem are found. In the rest of the paper, when we talk about the candidate set of a partial plan, we will be concerned about the subset of its candidates that belong to $\mathcal{K}$.

3.2 Candidate sets and Ground Linearizations

Traditionally, semantics for partial plans are given in terms of their ground linearizations (rather than in terms of candidate sets, as is done here).

Definition 12 *A* **ground linearization** *(aka completion) of a partial plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}, \mathcal{A}\rangle$ is a fully instantiated total ordering of the steps of $\mathcal{P}$ that is consistent with O (i.e., a topological sort) and $\mathcal{B}$.*

A ground linearization is said to be a **safe ground linearization** *if and only if it also satisfies all the auxiliary candidate constraints.*[7]

[7]Note that safe ground linearizations do not have to satisfy auxiliary *solution* constraints.

For the example plan discussed above, $t_0 t_1 t_2 t_\infty$ is the only ground linearization, and it is also a safe ground linearization. Safe ground linearizations are related to candidate sets in the following technical sense:

Proposition 3 *Every candidate S belonging to the candidate set of a partial plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}, \mathcal{A}\rangle$ is either a minimal candidate, in that it exactly matches a safe ground linearization of $\mathcal{P}$ (except for the dummy steps t_0 and t_∞, and modulo the mapping of $\mathcal{ST}$), or is a safe augmentation of a minimal candidate obtained by adding additional ground operators without violating any auxiliary candidate constraints.*

This proposition follows from the definition of candidate constraints. Consider any candidate (ground operator sequence) S of the plan $\mathcal{P}$. Let $\mathcal{M}$ be the candidate mapping according to which S satisfies the Definition 10. Consider the operator sequence S' obtained by removing from S every element $S[i]$ such that $\mathcal{M}$ does not map any step in $\mathcal{P}$ to $S[i]$. From the definition of candidate set, it is easy to see that S' must match with a ground linearization of $\mathcal{P}$. Further, since S satisfies all the auxiliary candidate constraints, and since candidate constraints are monotonic, it cannot be the case that S' violates them. Thus, S' matches a safe ground linearization of the plan.

For the example plan discussed above, $o_1 o_2$ is a minimal candidate because it exactly matches the safe ground linearization $t_0 t_1 t_2 t_\infty$, under the mapping $\mathcal{ST}$. The ground operator sequence $o_1 o_3 o_2 o_4$, where o_3 does not add or delete p, and o_4 does not add or delete q, is a candidate of this plan. It can be obtained by augmenting the minimal candidate $o_1 o_2$ with the ground operators o_3 and o_4 without violating auxiliary candidate constraints.

During search, it is often useful to recognize and prune inconsistent plans (as they clearly cannot lead to solutions). Proposition 4, which is a direct consequence of Proposition 3, provides a method of checking consistency in terms of safe ground linearizations:

Proposition 4 *A search node in refinement planning is consistent if and only if the corresponding partial plan has at least one safe ground linearization.*

4 A generalized algorithm for Refinement Planning

The algorithms `Find-plan` and `Refine-Plan` in Figure 2 instantiate the refinement search within the context of planning. In particular, they describe a generic refinement-planning algorithm, the specific instantiations of which cover the complete gamut of plan-space planners. Table 1 characterizes many of the well known plan-space planners as instantiations of the `Refine-Plan` algorithm. The algorithms are *modular* in that 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 ADL action representation [19].

The refinement process starts with the partial plan $\mathcal{P}_\emptyset$, which contains the steps t_0 and t_∞, and has its agenda and auxiliary solution constraints initialized to the top level

goals of attainment (preconditions of t_∞). The procedure `Refine-Plan` specifies the refinement operations done by the planning algorithm. Comparing this algorithm to the refinement search algorithm in Figure 1, we note that it uses two broad types of refinements: the establishment refinements (steps 2.1, 2.2); and the tractability refinements (step 3). In each refinement strategy, the added constraints include step addition, ordering addition, binding addition, as well as addition of auxiliary constraints. In the following, we briefly review the individual steps of these algorithms.

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 candidate from the candidate set of a partial plan. Since enumerating and checking the full candidate set can be prohibitively expensive, most planners concentrate instead on the safe-ground linearizations of the plan (which bound the candidate set from above; see Proposition 3), and see if any of those correspond to solution candidates. In particular, the following is the default solution constructor used by all existing refinement planners (with respect to which completeness results are proven):

Definition 13 (All-sol) *Given a partial plan $\mathcal{P}$, all-sol returns with success only when $\mathcal{P}$ is consistent, all of its ground linearizations are safe, and each safe ground linearization corresponds to a ground operator sequence that is a solution candidate of $\mathcal{P}$.*

The termination criteria of *all-sol* correspond closely to the notion of necessary correctness of a partially ordered plan, first introduced by Chapman [2]. Existing planning systems implement *All-sol* in two different ways: Planners such as Chapman's TWEAK [2] 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 [15] and UCPOP [22] depend on protection strategies and conflict resolution (see below) 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

The most fundamental refinement operation is the so-called establishment operation. It selects a precondition $\langle C, s \rangle$ of the plan (where C is a precondition of a step s), and refines (i.e., adds constraints to) the partial plan such that different steps act as contributors of C to s in different refinements. Chapman [2] and Pednault [19] provide theories of sound and complete establishment refinement. Pednault's theory is more general as it deals with actions containing conditional and quantified effects.[8] It is possible to limit `Refine-Plan` to establishment refinements alone and still get a sound and complete (in the sense of Definition 2) planner (using the default solution constructor *all-sol* described earlier).

In Pednault's theory, establishment of a condition c at a step s essentially involves selecting some step s' (either

[8] And also separates checking truth of a proposition from planning to make that proposition true, see [10].

Algorithm Find-Plan($\mathcal{I}, \mathcal{G}$) **Parameters**: `sol`: Solution constructor function.

1. Initialize the `open` list with the null plan $\mathcal{P}_\emptyset$: $\langle \{t_0, t_\infty\}, \{t_0 \prec t_\infty\}, \emptyset, \{t_0 \rightarrow \texttt{start}, t_\infty \rightarrow \texttt{fin}\}, \mathcal{L}_\emptyset, \mathcal{A}_\emptyset \rangle$, where corresponding to each goal $g_i \in G$, $\mathcal{A}_\emptyset$ contains $\langle g_i, t_\infty \rangle$, and $\mathcal{L}_\emptyset$ contains $\langle g_i @ t_\infty \rangle$.
2. Nondeterministically pick a partial plan $\mathcal{P}$ from `open`.
3. If `sol`$(\mathcal{P}, \mathcal{G})$ returns a solution, return it, and terminate. If it returns $*fail*$, skip to Step 2. If it returns $\perp$, call `Refine-plan`$(\mathcal{P})$ to generate refinements of $\mathcal{P}$. Add all the refinements to the `open` list; Go back to 2.

Algorithm Refine-Plan($\mathcal{P}$) /*Returns refinements of $\mathcal{P}$ */ **Parameters**: (i) `pick-prec`: the routine for picking the preconditions from the plan agenda for establishment. (ii) `interacts?`: the routine used by pre-ording to check if a pair of steps interact. (iii) `conflict-resolve`: the routine which resolves conflicts with auxiliary candidate constraints.

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

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

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

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

- **3.a. Pre-Ordering:** Impose additional orderings between every pair of steps of the partial plan that possibly interact according to the static interaction metric `interacts?`. *Backtrack point; all interaction orderings need to be considered.*
- **3.b. Conflict Resolution:** Add orderings, bindings and/or secondary (preservation) preconditions to resolve conflicts between the steps of the plan, and the plan's auxiliary candidate constraints. *Backtrack point; all possible conflict resolution constraints need to be considered.*

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

5. Return the refined partial plan (if it is not pruned).

Figure 2: *A generalized refinement algorithm for plan-space planning*

Planner	Soln. Constructor	Goal Selection	Book-keeping	Tractability Refinements
Tweak [2]	MTC-based ($O(n^4)$ for TWEAK rep; NP-hard with ADL)	MTC-based ($O(n^4)$ for TWEAK rep; NP-hard with ADL)	None	None
UA [16]	MTC-based $O(n^4)$	MTC-based $O(n^4)$	None	Unambiguous ordering
Nonlin [26]	MTC (Q&A) based	Arbitrary $O(1)$	Goal Protection via Q&A	Conflict Resolution
TOCL [1]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Total ordering
Pedestal [14]	Protection based $O(1)$	Arbitrary $O(1)$	Interval Protection	Total ordering
SNLP [15] UCPOP [22]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Conflict resolution
MP, MP-I [8]	Protection based	Arbitrary	(Multi) contributor protection	Conflict resolution
SNLP-UA (cf. Section 5.3.1)	Protection based $O(1)$/ MTC based/$O(n^4)$	Arbitrary $O(1)$/ Pick if nec. false. /$O(n^4)$	Contributor protection	Unambiguous Ordering

Table 1: Characterization of existing planners as instantiations of `Refine-Plan`

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 precondition preconditions are called secondary preconditions of the action. These are added to the agenda of the partial plan, and are treated in the same way as normal preconditions. (This includes adding a PTC $\langle c@s \rangle$ to the auxiliary solution constraints, whenever a precondition $\langle c, s \rangle$ is added to the agenda; see Section 3.1).

Goal Selection: The strategy used to select the particular precondition $\langle C, s \rangle$ to be established, (called goal selection strategy) can be arbitrary, can depend on some ranking based on precondition abstraction [24], and/or demand driven (e.g. select a goal only when it is not already necessarily true according to the modal truth criterion [2]). The last strategy, called MTC-based goal selection, involves reasoning about truth of a condition in a partially ordered plan, and can be intractable for general partial orderings consisting of ADL [19] actions (see Table 1, as well as the discussion of pre-ordering strategies in Section 4.5.1.).

4.3 Book Keeping and Protecting establishments

It is possible to do establishment refinement without book-keeping step. Chapman's TWEAK [2] is such a planner. However, such a planner is not guaranteed to respect its previous establishment decisions while making new ones, and thus may 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 repeatedly establishing and clobbering the same precondition. The book-keeping step attempts to reduce these types of redundancy.

At its simplest, the book-keeping may be nothing more than removing each precondition from the agenda of the partial plan once it is considered for establishment. When the agenda of a partial plan is empty, it can be pruned without loss of completeness (this is because the establishment refinement looks at all possible ways of establishing a condition at the time it is considered).

A more active form of book-keeping 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 auxiliary candidate constraints on the partial plan to record the establishment decisions, and ensuring that they are not violated by the later refinements. If they are violated, then 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 (aka causal link protection, or protection intervals), and contributor protection (aka exhaustive causal link protection [8]). 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 [14]), the book-keeping constraint requires that no candidate of the partial plan can have p *deleted* between operators corresponding to s' and s. It can thus be modeled in terms of interval preservation constraint $\langle s', p, s \rangle$. Finally, for book keeping based on contributor protection, the auxiliary constraint requires that no candidate of the partial plan can have p either *added* or deleted between operators corresponding to s' and s.[9] This contributor protection can be modeled in terms of the twin interval preservation constraints $\langle s', p, s \rangle$ and $\langle s', \neg p, s \rangle$.

While most planners use one or the other type of protection strategies exclusively for all conditions, planners like NONLIN and O-Plan [26, 27] post different book-keeping 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 (see [8] for a motivation and formal treatment of this idea).

While all the book-keeping strategies described above avoid considering same precondition for establishment more than once, only the contributor protection eliminates the redundancy of overlapping candidate sets, by making estab-

[9] See [7] for a coherent reconstruction of the ideas underlying goal protection strategies.

lishment refinement systematic. Specifically, we have:

Proposition 5 *Establishment refinement with exhaustive causal links 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 [15, 7]). To understand this, consider the following partial plan (where the agenda and the auxiliary solution constraints are omitted for simplicity):

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

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 provide the condition q. The establishment refinement generates two partial plans:

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

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

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.

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 completeness point of view, consistency check is an optional step.) Given the relation between the safe ground linearizations and candidate sets, 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 auxiliary candidate constraints (even for ground partial plans without any variables). Specifically, we have:

Proposition 6 *Given a partial plan whose auxiliary candidate constraints contain interval preservation constraints, checking if there exists a safe ground linearization of the plan is NP-hard.*

This proposition directly follows from the result in [25], which shows that checking whether there exists a conflict-free ground linearization of a partial plan with interval protection 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. Specifically, when checking the satisfiability of a set of constraints is intractable, we can still achieve polynomial refinement cost 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, while preserving the completeness and systematicity of the search. This is the primary motivation behind tractability refinements. There are two types of tractability refinements: *pre-ordering* and *conflict resolution*. Both these aim to maintain partial plans all of whose ground linearizations are safe ground linearizations.

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 auxiliary candidate constraints can be checked without explicitly enumerating all the ground linearizations. Two possible pre-ordering techniques are *total ordering* and *unambiguous ordering* [16]. 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.[10] 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. [16, 7]). 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 [16] for this purpose.

4.5.2 Conflict Resolution Refinements

Conflict resolution refines a given partial plan with the aim of compiling the auxiliary constraints into the ordering and binding constraints. Specifically, the partial plan is refined (by adding ordering, binding or secondary preconditions [19] to the plan) until each possible violation of the auxiliary candidate constraint (called conflict) is individually resolved. The definition of conflict depends upon the specific type

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

of auxiliary constraint. An interval preservation constraint $\langle s_i, p, s_j \rangle$ is violated (threatened) whenever a step s' can possibly come between s_i and s_j and not preserve p. 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 [19], to the plan agenda (and the corresponding PTCs to the auxiliary solution constraints; see Section 3.1). When all conflicts are resolved this way, the resulting refinements will have the property that all their ground linearizations are safe. Thus, checking the partial plan consistency will amount to checking for the existence of ground linearizations. This can be done by checking ordering and binding consistency.

5 Applications of the Unified Framework

The componential view of refinement planning, provided by the `Refine-Plan` algorithm has a variety of applications in understanding and analyzing the performance tradeoffs in the design of plan-space planning algorithms. I will briefly discuss these in this section.

5.1 Explication and analysis of Design Tradeoffs

We have seen that the various ways of instantiating `Refine-Plan` algorithm correspond to the various choices in designing the plan-space planning algorithms. The model for estimating search space size, developed in Section 2, provides a way of analyzing the search space size vs. refinement cost tradeoffs provided by these different design choices. Understanding these tradeoffs allows us to predict the circumstances under which specific techniques will lead to performance improvements.

If C is the average cost per invocation of the `Refine-Plan` algorithm, b is the average branching factor and d_e is the effective depth of the search, then the cost of the planning (in a breadth-first regime) is $C \times |\mathcal{F}_{d_e}|$ (where $\mathcal{F}_{d_e}$ is the size of the fringe at $d_e{}^{th}$ level of the search tree. From Section 3 (Eqn. 1), we have

$$\mathcal{F}_{d_e} = \frac{|\mathcal{K}| \times \rho_{d_e}}{\kappa_{d_e}} = b^{d_e}$$

C itself can be decomposed into three main components: $C = C_e + C_c + C_s$, where C_e, is the establishment cost (including the cost of selecting the open goal to work on), C_s is the cost of solution constructor, and C_c is the cost of consistency check. The average branching factor, b can be split into two components, b_e, the establishment branching factor, and b_t the tractability refinement branching factor, such that $b = b_e \times b_t$. b_e and b_t correspond, respectively, to the branching made in steps 2 and 3 of the `Refine-Plan` algorithm.

This simple model is remarkably good at explaining and predicting the tradeoffs offered by the different ways of instantiating `Refine-Plan` algorithm. One ubiquitous tradeoff is between that of search space size ($|\mathcal{F}_d|$) and refinement cost (C): *almost every method for reducing C increases $\mathcal{F}_{d_e}$ and vice versa*. For example, consider the MTC-based and protection-based solution constructors discussed in Section 4.1. Protection based constructors have to wait until each precondition of the plan has been considered for establishment explicitly, while the MTC-based constructors can terminate the search as soon as all the preconditions in the partial plan are necessarily correct (according to the modal truth criterion). MTC-based solution constructors can thus allow the search to end earlier, reducing the effective depth of the search, and thereby the size of the explored search space. In terms of candidate space view, such stronger solution constructors lead to larger κ_d at the termination fringe. However, at the same time they increase the cost of refinement C (specifically the C_s factor). For example, MTC-based solution constructor has to reason with all safe ground linearization of the plan explicitly, and can thus be intractable for general partial orderings involving ADL actions [10, 2]. Protection-based constructor, on the other hand need only check that the agenda is empty, and that there are no unresolved conflicts (which can be done in $O(1)$ time).

Book-keeping techniques aim to reduce the redundancy factor ρ_d. This tends to reduce the fringe size, $|\mathcal{F}_d|$. Book keeping constraints do however tend to increase the cost of consistency check. In particular, checking the consistency of a partial plan containing interval preservation constraints is NP-hard even for ground plans in TWEAK representation (c.f. [25]). Tractability refinements primarily aim to reduce the C_c component of refinement cost. In terms of search space size, tractability refinements further refine the plans coming out of the establishment stage, thus increasing the (b_t component of the) branching factor.

This $|\mathcal{F}_d|$ vs. C tradeoff also applies to other types of search-space reduction techniques such as deferment of conflict resolution [21, 7]. 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`. 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 by reducing the tractability branching factor b_t. This does not come without a penalty however. Specifically, when the planner does such partial conflict resolution, the consistency check has to once again test for existence of safe ground linearizations, rather than order and binding consistency (making consistency check intractable once again). Using weaker consistency checks, such as order and binding consistency check, can lead to refinement of inconsistent plans, thereby reducing κ_d and increasing $|\mathcal{F}_d|$.

5.1.1 Depth First Search Regimes

Although the above analysis dealt with breadth-first search regimes, the `Refine-Plan` algorithm also allows us to analyze the performance of different planning algorithms in depth first regimes [7]. Here, the critical factor in estimating the explored search space size is the probability that the planner picks a refinement that contains at least one solution candidate. Even small changes in this probability, which we shall call *success probability*, can have dramatic effects on performance.

To illustrate, let us consider the effect of tractability refinements on the success probability. If we approximate the behavior of all the refinement strategies used by

Refine-Plan as random partitioning of candidate set of a plan into some number of children nodes, then it is possible to provide a quantitative estimate of the success probability. Consider the refinement of a plan $\mathcal{P}$ by Refine-Plan. Suppose that $\mathcal{P}$ has m solution candidates in its candidate set. If b is the average branching factor, then Refine-Plan splits $\mathcal{P}$ into b different children plans. The success probability is just the probability that a random node picked from these b new nodes contains at least one solution candidate. This is just equal to $q(m, b)$ where $q(m, b)$ is the binomial distribution:[11]

$$q(m,b) = \sum_{i=0}^{m-1} \binom{m}{b} \frac{1}{b^{m-i}} \left(1 - \frac{1}{b}\right)^{i}$$

It can be easily verified that for fixed m, $q(m, b)$, the success probability, monotonically decreases with increasing b. As the success probability reduces, the size of the explored search space increases. Thus, under random partitioning model, the addition of tractability refinements tends to increase the explored search space size even in depth-first search regimes. The only time we will expect reduction in search space size is if the added refinements distribute the solutions in a non-uniform fashion, thereby changing the apparent solution density (c.f. [17]).

5.2 Facilitation of Well-founded Empirical Comparisons

Given the variety of ways in which Refine-Plan can be instantiated, it is important to understand the comparative advantages of the various instantiations. While theoretical analyses of the comparative performances are desirable, sometimes either they are not feasible, or the performance tradeoffs may be critically linked to problem distributions. In such cases, comparisons must inevitably be based on empirical studies.

The unified framework offers help in designing focused empirical studies. In the past, empirical analyses tended to focus on a wholistic ''black-box'' comparisons of brand-name planning algorithms, such as TWEAK vs. SNLP (c.f. [13]). It is hard to draw meaningful conclusions from such comparisons, since when seen as instantiations of our Refine-Plan algorithm, they differ on a variety of dimensions (see Table 1). A more meaningful approach, facilitated by the unifying framework of this paper, involves comparing instantiations of Refine-Plan that differ only on a single dimension. For example, if our objective is to judge the utility of specific protection (book-keeping) strategies, we could keep everything else constant and vary only the book-keeping step in Refine-Plan. In contrast, when we compare TWEAK [2] with SNLP [15], we are not only varying the protection strategies, but also the goal selection, conflict resolution and termination (solution constructor) strategies, making it difficult to form meaningful hypotheses from empirical results.

In [11], I exploit this experimental methodology to compare the empirical performance of a variety of normalized instantiations of Refine-Plan algorithm. These experiments reveal that the most important cause for the performance differentials among different refinement planners are the differences in the tractability refinements they employ. Although tractability refinements increase the b_t component of the branching factor, they may also indirectly lead to a reduction in the establishment branching factor, b_e. The overall performance of the planner thus depends on the interplay between these two influences. The book-keeping (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.

5.3 Designing planners with better tradeoffs

By providing a componential view of the plan-space planning algorithms, and explicating the spectrum of possible planning algorithms, the unified framework also facilitates the design of novel planning algorithms with interesting performance tradeoffs. We will look at two examples briefly:

5.3.1 Strong systematicity with polynomial refinement

As we noted earlier, a refinement search is strongly systematic if it is systematic, and never refines an inconsistent node. From Table 1, we see that there exist no partial order planning algorithms which are both strongly systematic and have polynomial time refinement complexity. SNLP, which uses contributor protection, is systematic and can be strongly systematic as long as the consistency check is powerful enough to remove every inconsistent plan from search. However, checking whether a general partially ordered plan is consistent with respect to a set of exhaustive causal links is NP-hard in general [25]. This raises the interesting question: *Is it possible to write a* partial order *planning algorithm that is both strongly systematic and has a polynomial time refinement cycle?*

Our modular framework makes it easy to synthesize such an algorithm. Table 1 describes a novel planning algorithm called SNLP-UA which uses exhaustive causal links for book-keeping, and uses a pre-ordering refinement whereby every pair of steps s_1 and s_2 such that an effect of s_1 possibly codesignates with a precondition or an effect of s_2, are ordered with respect to each other.[12] Such an ordering converts all potential conflicts into either necessary conflicts, or necessary non-conflicts.[13] This in turn implies that either all ground linearizations are safe or none of them are. In either case, consistency can be checked in polynomial time by examining any one of the ground linearizations. SNLP-UA is thus strongly systematic, maintains partially ordered plans, but still keeps the refinement cost polynomial. It could thus strike a good balance between systematic planners such as SNLP and unsystematic, but polynomial-time refinement

[11] $q(m, n)$ is the probability that a randomly chosen urn will contain at least one ball, when m balls are independently randomly distributed into n urns. This is equal to probability that a randomly chosen urn will have all m balls plus the probability that it will have $m - 1$ balls and so on plus the probability that it will have 1 ball.

[12] Note that this definition of interaction is more general than the one used by UA [17]. It is required because of the contributor protections used by SNLP-UA (see [7]).

[13] For actions with conditional effects, a necessary conflict can be *confronted* by planning to make the preservation preconditions true for the interacting step.

planners such as UA. In [11], I provide empirical comparisons between SNLP-UA and other possible instantiations of Refine-Plan.

5.3.2 Polynomial eager solution-constructors

As discussed in Section 4.1, *all-sol*, the solution constructor used in all existing plan-space planners returns with success only when *all* the safe ground linearizations of the partial plan are solutions. Our refinement search paradigm suggests that such solution constructors are over-conservative since the goal of planning is only to find *one* solution. In contrast, *eager* solution constructors, that stop as soon as they find a safe ground linearization that is a solution, will reduce solution depth, increase κ, and there by reduce search-space size. The most eager constructor, which I call *all-eager-constructor*, would stop as soon as the partial plan contains at least *one* safe ground linearization that is a solution. Unfortunately both the *all-sol* and *all-eager-constructor* are NP-hard in general, as the problem of finding necessary and possible truth of a proposition in a partially ordered plan can respectively be reduced to them [10]. This raises the interesting question: *Are there any domain-independent eager solution-constructors that are tractable?* I answer the question in the affirmative by providing a family of tractable eager solution constructors called *k-eager-constructors*:

Definition 14 (k-eager Constructor) *Given a partial plan $\mathcal{P}$, a k-eager-constructor randomly enumerates at most k ground linearizations of $\mathcal{P}$, and returns any one of them that is safe and corresponds to a solution for $\mathcal{P}$.*

The *k-eager-constructors* are tractable since they only enumerate and check at most k different ground linearizations. Based on the value of k, they define a family of solution constructors whose cost increases and effective solution depth reduces with increasing k. Finally, the solution depth of *k-eager-constructor* is guaranteed to lie between that of *all-eager-constructor* and the MTC-based *all-sol* solution constructor, thus providing an interesting balance between the two. Empirical studies are currently under way to assess the practical impact of these constructors.

5.4 Pedagogical explanatory power

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 book-keeping 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, in the past it has been erroneously claimed (e.g. [13]) that the systematicity of SNLP *increases* the effective 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 book-keeping 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 exhaustive causal links 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 [17] and causal-link based planners such as SNLP [15], it also suggests fruitful ways of integrating the ideas in the two planning techniques (cf. SNLP-UA in Section 5.3.1).

6 Extending the framework

In this section, I will discuss how the Refine-Plan framework can be extended to handle a wider variety of behavioral constraints (beyond goals of attainment), as well as other types of planning models.

Maintenance goals are a form of behavioral constraints which demand that a particular condition be maintained (not violated) throughout the execution of the plan (e.g. keep A on B while transferring C to D; avoid collisions while traveling to room R). They can be modeled in the Refine-Plan algorithm simply as auxiliary candidate constraints. For example, we can maintain $On(A, B)$ by adding the interval preservation constraint $\langle t_0, On(A, B), t_\infty \rangle$ to $\mathcal{P}_\emptyset$ in the Find-Plan algorithm in Figure 2.

Intermediate goals are useful to describe planning problems which cannot be defined in terms of the goal state alone. As an example, consider the goal of making a round trip from Phoenix to San Francisco.[14] Since the initial and final location of the agent is Phoenix, this goal cannot be modeled as a goal of attainment, i.e., a precondition of t_∞ (unless time is modeled explicitly in the action representation [23]). However, we can deal with this goal by adding an additional dummy step (say t_D) to the plan such that t_D has a precondition $At(Phoenix)$ and t_∞ has a precondition $At(SFO)$, and $t_0 \prec t_D \prec t_\infty$.

Many refinement planners (especially the so-called task reduction planners) use extensions such as condition-typing [26], time-windows [27] and resource based reasoning [27, 28]. Many of these extensions can be covered with the auxiliary constraint mechanism. Time windows and resource reasoning aim to prune partial plans that are infeasible in terms of their temporal constraints and resource requirements. These can, in principle, be modeled in terms of monotonic auxiliary constraints. Condition typing allows the domain user to specify how various preconditions of an operator should be treated during planning [26]. In particular, some planners use the notion of *filter conditions*, which are the applicability conditions of the operators that should never be explicitly considered for establishment. Filter conditions thus provide a way for the domain writer to *disallow* certain types of solutions (e.g., building an airport in a city for the express purpose of going from there to another city) even if they satisfy the standard definition of plan solutions

[14] In the past, some researchers (e.g. [4]) have claimed (mistakenly) that intermediate goals of this type cannot be modeled in classical planning without hierarchical task reduction.

(see Definition 5). Filter conditions can be modeled as point truth constraints, and included in the auxiliary solution constraints (without adding them to the agenda) [12]. Since they are (non-monotonic) solution constraints, they cannot be used to prune partial plans. However, they can be used as a basis for selection heuristics (viz., to prefer partial plans which have already satisfied filter conditions).[15]

Finally, `Refine-Plan` can also be extended to cover planning models other than plan-space planning. To cover state-space planners (cf. [1]), we need to allow `Refine-Plan` to use incomplete establishment refinements, and backtrack over goal-selection to make the overall search complete. The HTN planners (cf. [27, 4]) can be modeled by extending the refinement algorithm such that its main refinement operation is task reduction rather than establishment (with establishment refinement being a particular way of reducing tasks); see [12].

7 Conclusion

In this paper, I have shown that refinement search provides a unifying framework for understanding the performance tradeoffs in plan-space planning. I have developed a formalization of plan-space planning in terms of refinement search, and gave a generic refinement search algorithm in which the complete gamut of plan-space planners can be cast. I have shown that this unifying framework facilitates explication and analysis of performance tradeoffs across a variety of planning algorithms. I have also shown that it could help in designing new algorithms with better cost-benefit ratios. Although I concentrated on the plan-space planners solving problems involving goals of attainment, I have shown (Section 6) that the framework can be extended to cover richer types of behavioral constraints, as well as other types of planners (e.g. state-space planners, hierarchical planners).

References

[1] A. Barrett and D. Weld. Partial Order Planning: Evaluating Possible Efficiency Gains. CSE TR 92-05-01, University of Washington, June 1992.

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

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

[4] 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.

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

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

[7] 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.[16]

[8] S. Kambhampati. Multi-Contributor Causal Structures for Planning: A Formalization and Evaluation. Arizona State University Technical Report, CS TR-92-019, July 1992. (To appear in *Artificial Intelligence*. A preliminary version appears in the Proc. of First Intl. Conf. on AI Planning Systems, 1992).

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

[10] S. Kambhampati and D.S. Nau. On the Nature and Role of Modal Truth Criteria in Planning. Tech. Report. ISR-TR-93-30, University of Maryland, March, 1993.

[11] S. Kambhampati. Design Tradeoffs in Partial Order (Plan Space) Planning. Submitted to AIPS-94 and AAAI-94.

[12] S. Kambhampati. HTN Planning: What? Why? and When? ASU Technical Report in preparation, 1994.

[13] 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.

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

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

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

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

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

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

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

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

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

[23] J.S. Penberthy. Planning with continuous change. Ph.D. Thesis. CS-TR 93-12-01. University of Washington. 1993.

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

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

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

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

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

[15] Some researchers [3] have suggested that filter conditions cannot be used in partial order planning without loss of completeness. I believe that this confusion is mainly a result of seeing filter conditions as filtering out refinement possibilities, as against solutions.

[16] Technical reports available via anonymous ftp from `enws318.eas.asu.edu:pub/rao`