Planning as Refinement Search: A unified framework for comparative analysis of Search Space Size and Performance

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

ASU CSE Technical Report 93-004
June, 1993

Abstract

In spite of the long history of classical planning, there has been very little comparative analysis of the search space characteristics of the multitude of existing planning algorithms. This has seriously inhibited efforts to fruitfully intergrate various approaches. In this paper we show that viewing planning as a general refinement search provides a unified framework for comparing the search spaces of various planning strategies, and in predicting their performance. We will provide a generic refinement search algorithm for planning, and show that all planners that search in the space of plans are special cases of this algorithm. In this process, we will provide a rational reconstruction of main ideas of refinement planning algorithms. We will then develop a model for estimating search space size of a refinement planner, and use this model to analyze a variety of tradeoffs between search space size, refinement cost and performance in refinement planning.

[*]This research is supported in part by National Science Foundation under grant IRI-9210997, and ARPA/Rome Laboratory planning initiative under grant F30602-93-C-0039.

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, [12, 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 of planning algorithms. There has been very little comparative analysis of the search space characteristics of the multitude of existing classical planning algorithms. There is a considerable amount of disagreement and confusion on the role and utility of even such long-standing concepts as ‘‘goal protection’’, and ‘‘protection intervals’’ -- not to mention the more recent ideas such as ‘‘systematicity’’ -- on the search space characteristics of planning algorithms. One 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, which in turn has severely inhibited fruitful integration of competing approaches.[1]

In this paper, we shall show that viewing planning as a refinement search provides a unified framework within which a variety of refinement planning algorithms can be effectively compared. We will start by characterizing planning as a refinement search, and provide semantics for partial plans and plan refinement operations. We will then provide a generic refinement planning algorithm in terms of which the whole gamut of the so-called plan-space planners, which search in the space of partial plans, can be expressed. We will use this unifying model to provide a rational reconstruction of main ideas of refinement planning algorithms. This reconstruction facilitates separation of important ideas underlying individual algorithms from ‘‘brand-names’’, and thus provides a rational basis for fruitfully integrating the various approaches. Finally, we develop a model for estimating the search space size of a refinement planner, and use it to analyze the tradeoffs provided by many approaches for improving performance by reducing search space size. As our model does not assume any restrictions on action representations, it also facilitates evaluation of these approaches terms of their ability to scale up to more expressive action representations.

This paper is organized as follows: Section 2 reviews the objectives of classical planning, and defines the notions of solutions and completeness. Section 3 introduces general refinement search, and characterizes planning as a refinement search. It also provides semantics for partial plans in terms of refinement search, and develops a generic algorithm for refinement planning. In Section 4, this generic algorithm is used as a backdrop to provide a rational reconstruction of the main ideas of existing refinement planners. Particular attention is paid to the variety of techniques

[1]The work of Barrett and Weld [24] as well as Minton et. al. [15, 16] 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.

used by planners to avoid looping, redundancy and non-minimal plans. Section 5 provides a qualitative model for estimating search space size of refinement planners, and uses it to analyze the tradeoffs provided by a variety of approaches for improving performance through search space reduction, including ‘‘systematicity’’, ‘‘strong systematicity’’, and ‘‘conflict deferment’’. Section 6 summarizes the contributions of the paper and discusses potential applications of the framework for refinement planning developed in this paper. Related work is discussed throughout the paper at appropriate places.

2 Goals, Solutions and Completeness in classical planning

Whatever the exact nature of the planner the ultimate aim of (classical) planning is to find a *ground operator sequence*, which when executed will produce desired *behaviors* or sequences of world states. Thus, goals of planning should be seen as specifications on the desired behaviors (e.g. as predicates on state sequences). A planning problem is thus specified by providing the description of the initial state, and a specification of behavioral constraints. Most classical planning techniques have traditionally concentrated on the sub-class of goals called the goals of attainment, which essentially constrain the agent’s attention to behaviors that end in world states satisfying desired properties. Unless otherwise stated, this is the class of goals we shall also be considering in this paper.

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 STRIPS representation with *Add*, *Delete* and *Precondition* formulas. The subset of this representation where all three formulas can be represented as conjunctions of function-less first order literals, and all the variables have infinite domains, will be referred to as TWEAK representation (c.f. [1, 9]).

Given this vocabulary, a solution for a planning problem can be formally defined as:

Definition 1 (Solution) *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 (i) if S is executable, i.e.,*

$$\begin{aligned}
\mathcal{I} &\vdash prec(o_1) \\
o_1(\mathcal{I}) &\vdash prec(o_2) \\
&\cdots \\
o_{n-1}(o_{n-2} \cdots (o_1(\mathcal{I}))) &\vdash prec(o_n)
\end{aligned}$$

(where $prec(o)$ denotes the precondition formula of the operator o) and (ii) the sequence of states $\mathcal{I}, o_1(\mathcal{I}), \cdots, o_n(o_{n-i} \cdots (o_1(\mathcal{I})))$ satisfies the behavioral constraints specified in the planning

goals. For goals of attainment, this latter requirement is stated solely in terms of the last state resulting from the plan execution:

$$o_n(o_{n-i}\cdots(o_1(\mathcal{I}))) \vdash \mathcal{G}$$

The definition of solution, given above, admits many redundant operator sequences as solutions. Suppose we are interested in goals of attainment, and the operator sequence S is a solution. From S it is possible to derive potentially infinite number of other solutions by adding additional operators which are executable, and do not interact with the existing operators of S. The presence of such redundant solutions necessitates focus on minimal solutions:

Definition 2 (Minimal Solutions) *A solution S is said to be minimal,*[2] *if no operator sequence obtained by removing some of the steps from S is also a solution.*

A specific type of non-minimal solutions, called ''bloated solutions,'' are of particular interest (see below):

Definition 3 (Bloated Solutions) *A solution S is said to be bloated (c.f. [12]) if the operator sequence S' obtained by deleting some operator $o \in S$, is also a solution. A solution that is not bloated is called an un-bloated solution.*

(Note that un-bloated solutions are not necessarily minimal; it may be possible to remove a non-singleton set of steps from them, while still keeping them as a solution.) Completeness results for classical planners are typically stated and proved with respect to some class of minimal solutions.[3]:

Definition 4 (Completeness) *A planner is said to be complete if it can find all minimal solutions for every solvable problem.*

Many planners described in the literature are in fact complete for the larger class of unbloated solutions [23, 12]. Despite the statement of completeness results in terms of minimal plans, the search spaces of most classical planners do however contain bloated solutions; the extent of bloated plan generation does vary from planner to planner. We will see that the desire to avoid planning decisions that lead to non-minimal solutions, is one of the implicit driving forces behind the development of many important ideas in planning algorithms.

[2]Notice that minimal solutions are defined with respect to the solution criterion. For example, in most classical planning systems dealing with goals of attainment, any solution which gives rise to state-cycles (e.g. the sequence of actions Open the door, Close the door one after other lead us back to the same state) are non-minimal. However, if the goal of the planner itself was to exhibit a behavior (state sequence) containing the same state more than once, they may not be non-minimal.

[3]The definition of completeness with respect to *all* rather than *a* minimal solution is in a way ironic considering that most practical planners require only one solution for a given problem, and optimality of solutions has not been a primary consideration in classical planning.

3 Planning as Refinement Search

Almost all classical planners that search in the space of plans use *refinement search* to navigate the space of ground operator sequences and find a solution for the given planning problem. A refinement search (or split-and-prune search [19]) can be visualized either as a process of starting with a partial solution and adding detail to the partial solutions until it satisfies the solution constraints, or equivalently, as a process of starting with the set of all potential candidates, and splitting the set repeatedly until a solution candidate can be picked up from one of the sets in bounded time.[4] The set of all candidates for the refinement search is denoted by $\Im$. The denotation of a search node $\mathcal{N}$ in refinement search, written as $[[\mathcal{N}]]$, is a subset of $\Im$. The denotation of the initial search node is the set of all candidates, $\Im$.

In the case of planning, $\Im$ is the set of all ground operator sequences. Suppose the domain contains three ground actions $a1$, $a2$ and $a3$, then the regular expression $\{a1|a2|a3\}^*$ would describe the potential solution space for this domain. Each search node corresponds to some subset of these ground operator sequences.

Although it is conceptually simple to think of search nodes in refinement search in terms of their denotations (i.e, sets of potential candidates), we obviously do not want to represent the candidate sets explicitly in our implementations. Instead, the candidate set is typically implicitly represented as a generalized constraint set (c.f. [5] such that every potential candidate that is *consistent* with the constraints in that set is taken to belong to the candidate set of the search node. (Because of this, search nodes in refinement search are often seen also as ‘‘partial solutions’’.)

3.1 Semantics of Partial Plans

In the case of planning, search nodes can be seen as a set of constraints defining partial plans (in the following, we will be using the terms ‘‘search node’’ and ‘‘partial plan’’ interchangeably). At an abstract level, the search-node (partial plan) representation used by all refinement planners can be described in terms of a 5-tuple:

$$\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$$

where:

- T is the set of actions (step-names) in the plan; T contains two distinguished step names t_I and t_G.

- $\mathcal{ST}$ is a symbol table, which maps step names to domain operators. It always maps the special step t_I to the dummy operator `start`, and t_G to the dummy operator `fin`. The

[4]Pearl defines termination conditions of a split-and-prune strategy in a slightly different manner, requiring that the the set be split repeatedly until a singleton set containing a solution candidate is reached. The termination condition discussed here is more general-- search terminates as soon as a set of candidates is generated from which the solution constructor function can pick a solution candidate. It allows the complexity to be shared between the solution constructor function and the children generator. (This idea was suggested by David McAllester).

effects of `start` and the preconditions of `fin` correspond to the initial and final states of the plan, respectively.

- O is a partial ordering relation over T. For the special case where O defines a total ordering on the steps of T, $\mathcal{P}$ is called a totally ordered 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 on the orderings and bindings among the steps. Example of an auxiliary constraint that we shall see latter (Section 4.2.4)is a *exhaustive causal link*:[5]

$$\{s\} \xrightarrow{p} w \text{ where } s, w \in T \text{ and } p \in \text{preconditions}(w)$$

This constraint is interpreted as:

> ''s comes before w and gives p to w. No other step in the plan is allowed to come between s and w and add or delete p.''

We shall now formally define the candidate set of a partial plan:

Definition 5 (Candidate set of a Partial plan) *Given a search node $\mathcal{N}$ with the partial plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$, a ground operator sequence S is said to belong to $[[\mathcal{N}]]$ if and only if there exists a partial mapping between the operators of S and steps of $\mathcal{P}$ such that S satisfies all the constraints of $\mathcal{P}$. That is, S contains the operators corresponding to all the steps of $\mathcal{P}$ (excepting the dummy steps* `start` *and* `finish`*) in an ordering that is consistent with O and bindings consistent with $\mathcal{B}$, and S also satisfies the constraints of $\mathcal{L}$.*[6]

As an example, suppose the partial plan of the node $\mathcal{N}$ is given by the constraint set

$$\left\langle \begin{array}{l} \{t_I, t_1, t_2, t_G\}, \{t_I \prec t_1, t_1 \prec t_2, t_2 \prec t_G\}, \emptyset, \\ \{t_1 \rightarrow o_1, t_2 \rightarrow o_2, t_I \rightarrow \texttt{start}, t_G \rightarrow \texttt{finish}\}, \emptyset \end{array} \right\rangle$$

The ground operator sequence o_1, o_3, o_2 is consistent with the constraints of the partial plan, and thus belongs to the candidate set of $\mathcal{N}$. If the partial plan contains an exhaustive causal link (see above) as an auxiliary constraint:

[5]Auxiliary constraints can be more expressive than this. For example, an auxiliary constraint may express the constraint that a particular step must always be in a specific absolute position in every candidate, or that a particular operator must not come more than once in any candidate. The former can be used to model state based planners that allow step additions only at either end of the partial plan. The latter may be useful when dealing with planning situations where not all goals are goals of attainment.

[6]It is helpful to think of $\mathcal{L}$ as a filter on the candidates that satisfy the other constraints of $\mathcal{P}$.

$$\left\langle \begin{array}{l} \{t_I, t_1, t_2, t_G\}, \{t_I \prec t_1, t_1 \prec t_2, t_2 \prec t_G\}, \emptyset, \\ \{t_1 \rightarrow o_1, t_2 \rightarrow o_2, t_I \rightarrow \texttt{start}, t_G \rightarrow \texttt{finish}\}, \{\{t_1\} \xrightarrow{P} t_2, \{t_2\} \xrightarrow{Q} t_G\} \end{array} \right\rangle$$

Then, candidate (ground operator sequence) o_1, o_3, o_2 is consistent with the constraints of the partial plan, and thus belongs to the candidate set of $\mathcal{N}$ if and only if o_3 does not have P as an add or delete literal. Similarly, the candidate o_1, o_2, o_5 belongs to the candidate set of $\mathcal{N}$ if and only if o_5 does not add or delete Q.

Partial order vs. Total order planning: The only difference between partial order and total order planners are the extent of ordering in the partial plans. In total order planners, the partial plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ of every search node is totally ordered (i.e., O defines a total ordering on T), while in partial order planners it may be partially or totally ordered. The primary motivation behind partial order planning is efficiency -- by deferring orderings among operators, partial order planners avoid premature commitment and the consequent unnecessary backtracking, thereby improving performance [24, 9]. It is important to note that while partial order and total order planners use different representation constraints on the partial plans, the main goal of planning in either case still remains finding a ground operator sequence that is executable and satisfies goal criteria. In particular, partial order planners do not aim to find least constrained partially ordered operator sequences. It is easy to see that a total ordering places more constraints on the partial plan than a partial ordering. Thus, all else being equal, the difference between a total ordering planner and a corresponding partial ordering planner is merely that the candidate sets (denotations) of the the former are on the average smaller than that of the latter. As we shall see latter, this tends to reduce the overall size of search space of partial order planners, while at the same time increasing the cost of refinement in partial order planning. We will also see that from a refinement search point of view, partial order planning is more natural than total order planning. The motivations for the latter come in terms of the refinement cost.

Ground Linearizations and Candidate sets: As mentioned earlier, partial plans can also be seen as partial solutions to the planning problems. Previous formalizations of refinement planning, especially that of Chapman's TWEAK, took this approach and discussed notions of correctness of partial plans. At this point, it is useful to introduce some of that terminology and relate it to candidate set based semantics of partial plans.

Definition 6 (Ground Linearizations and Safe Ground Linearizations) *A ground linearization (aka completion) of a partial plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ is a fully instantiated total ordering of the steps of $\mathcal{P}$ that is consistent with O (i.e., a topological sort) and $\mathcal{B}$.*

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

For the example plan

$$\left\langle \begin{array}{l} \{t_I, t_1, t_2, t_G\}, \{t_I \prec t_1, t_1 \prec t_2, t_2 \prec t_G\}, \emptyset, \\ \{t_1 \to o_1, t_2 \to o_2, t_I \to \texttt{start}, t_G \to \texttt{finish}\}, \{\{t_1\} \xrightarrow{P} t_2, \{t_2\} \xrightarrow{Q} t_G\} \end{array} \right\rangle$$

discussed above, $t_I t_1 t_2 t_G$ is the only ground linearization, and it is also a safe ground linearization. Note that safe ground linearizations are related to *minimal* candidates in the following sense:

Proposition 1 (Candidate Sets and Safe Ground Linearizations) *Every candidate S belonging to the candidate set of a node $\mathcal{N} : \mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ is either a minimal candidate, in that it exactly matches a safe ground linearization of $\mathcal{N}$ (except for the dummy steps t_I and t_G, 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 constraints.*

Once again, for our example plan above, $o_1 o_2$ is a minimal candidate because it exactly matches the safe ground linearization $t_I t_1 t_2 t_G$, 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 constraints.

We will now define the notion of consistency of a partial plan/search node:

Definition 7 (Consistency of search nodes) *A search node is consistent if it has a non-empty candidate set (or alternately, the set of constraints in the search nodes are satisfiable).*

Nodes with empty candidate sets obviously cannot lead to a solution candidate and thus must be pruned. Given the relation between safe ground linearizations and candidate sets of a partial plan, it follows that:

Proposition 2 *A search node in refinement planning is consistent if and only if the corresponding partial plan has at least one safe ground linearization.*

We will also define a related planning-specific notion called *safety of partial plans*, which will be useful in discussing conflict resolution strategies used by many refinement planners:

Definition 8 (Safety of partial plans) *A partial plan is said to be safe if all its ground linearizations are safe*

It is easy to see that safety of partial plans is a stronger constraint than consistency.

Correctness of conditions w.r.t. ground linearizations: For the special case of goals of attainment, the correctness of a partial plan can be defined solely in terms of establishment of the preconditions of all the steps in the plan. A precondition of a partial plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ is a two tuple $\langle C, n \rangle$, where $n \in T$, and C is a precondition of n. $\langle C, n \rangle$ is said to be true in a ground linearization of $\mathcal{P}$, if there is a step n' which comes before n in the total ordering and n has an effect C, and no step between n' and n deletes C. If $\langle C, n\rangle$ is not true, then it is false in that ground linearization. $\langle C, n\rangle$ is said to be *necessarily true* in $\mathcal{P}$ if it is true in every ground linearization of $\mathcal{P}$. It is said to be *necessarily false* if it is false in every ground linearization of $\mathcal{P}$. In many instances $\langle C, n\rangle$ is neither necessarily true, nor necessarily false; in this case we call it *possibly true*[7] (i.e., it is true in some ground linearizations and false in others).

Chapman's definition of correctness of partial plans can be cast in our terminology as:

Definition 9 (Partial plan correctness) *A partial plan is said to be correct if and only if each of its ground linearizations exactly matches a solution candidate for the problem (modulo the mapping in $\mathcal{ST}$).*

3.2 A generic algorithm template for refinement planning

A refinement search is specified by providing a set of refinement operators (strategies) **R**, and a constructor function `sol`. The search process starts with the initial node ϕ, and consists of generating children nodes by application of refinement operators. A refinement operator (strategy) $\mathcal{R} \in$ **R** is a function that maps subsets of $\Im$ (the set of all candidates) to sets of subsets of $\Im$, such that if a subset $s \subset \Im$ is mapped to the set of subsets $\{s_1, s_2 \cdots s_n\}$, then $\forall_i \; s_i \subseteq s$. Refinement operators can thus 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 $\forall i \; [[\mathcal{N}'_i]] \subseteq [[\mathcal{N}]]$. Another example of an auxiliary constraint may be that a particular operator must not come more than once in any candidate.

Definition 10 (Completeness and Systematicity of a Refinement Strategy) *Let* **R** *be a refiment strategy that maps a node $\mathcal{N}$ to a set of children nodes $\{\mathcal{N}'_i\}$.* **R** *is said to be* **complete** *if $\bigcup_i [[\mathcal{N}'_i]] \equiv [[\mathcal{N}]]$ (i.e., no candidate is lost in the process of refinement).*

R *is said to be* **systematic** *if $\forall_{\mathcal{N}'_i, \mathcal{N}'_j} \; [[\mathcal{N}'_i]] \cap [[\mathcal{N}'_j]] \equiv \emptyset$.*

The search process involves selecting a candidate set and refining it (splitting it) into subsets. The search terminates when a node n is found for which the constructor function returns a solution candidate. A constructor function (aka solution constructor function) `sol` is a 2-place function which takes a search node and a solution criterion as arguments. It may return either one of three values:

1. *fail*, meaning that no candidate in $[[\mathcal{N}]]$ satisfies solution criterion

2. Some candidate $c \in [[\mathcal{N}]]$ which satisfies the solution criterion (i.e., c is a solution candidate)

[7]Note that the notion of "truth" here is conditional on executability. Just because a condition $\langle C, n\rangle$ is necessarily true, it doesn't mean that some ground linearization of $\mathcal{P}$ can be executed to produce C before n. See [9] for further discussion on this distinction.

```
Algorithm: Refinement Search(sol, $\mathbf{R}$)
1.  Initialize Search queue with $\phi$, the node with null constraint set
    Begin Loop
2.  If search queue is empty, terminate with failure
3.    Else, non-deterministically pick a node $n$ from the search queue
4.          If sol($n$,$\mathcal{G}$) returns a candidate $c$, return it with success
5.          Else, choose some refinement operator $\mathcal{R} \in \mathbf{R}$,
                  (Not a backtrack point.)
6.          generate $\mathcal{R}(n)$, the refinements of $n$ with respect to $\mathcal{R}$.
7.          Prune any nodes in $\mathcal{R}(n)$ that are inconsistent.
8.          Add the remaining nodes in $\mathcal{R}(n)$ to the search queue.
    End Loop
```

Figure 1: A generic Refinement search strategy

3. $\perp$, meaning that `sol` can neither construct a solution candidate, nor determine that no such candidate exists.

In the first case, $\mathcal{N}$ will be pruned. In the second case search terminates with success, and in the third, $\mathcal{N}$ will have to be refined further.

$\mathcal{N}$ is called a **solution node** if the call `sol`$(\mathcal{N}, \mathcal{G})$ returns a solution candidate. Given these definitions, the algorithm template in Figure 1 describes the general refinement search strategy[8]. A refinement search is complete, in the technical sense that it will find a solution candidate eventually if one exists, as long as all of its refinement strategies are complete (see above).

The procedure `Find-plan` in Figure 2 and the procedure `Refine-Plan` in Figure 3 instantiate the refinement search within the context of planning. In particular, they describe a generic refiment-planning algorithm, which can be specific instantiations of which cover the complete gamut of refinement planners, including partial ordering, total ordering, unambiguous,

[8]While there are several similarities between this algorithm and classical branch and bound search algorithm [17], there is one important difference worth noting: Branch and bound search typically assumes that all the candidates are solutions, and searches for an optimal solution candidate. Thus, the pruning part in line 7 of the refinement search algorithm in Figure 1 is similar to, but weaker than the pruning criteria used in branch and bound algorithms [17]. By replacing line 7 with a pruning criterion that prunes nodes such that the cost of the best possible solution candidate in their denotation is provably larger than the cost of best possible solution candidates in other nodes, we get branch and bound algorithm.

causal-link based, systematic, etc. The procedures are modular in that individual steps can be analyzed and instantiated relatively independently. Furthermore, the algorithm itself does not assume any specific restrictions on action representation. The procedures thus provide a uniform basis for understanding and comparing various planning algorithms in terms of search space size and performance.

The procedure `Refine-Plan` specifies the refinement operations done by the planning algorithm. The goal selection step picks a goal to work on next. The establishment step enumerates all possible ways of establishing the selected goal and generates one refinement (partial plan) for each establishment choice. The book keeping step adds auxiliary constraints to each partial plan, to avoid violating this establishment decision in latter refinements. The consistency check step checks to see if each refinement (partial plan) is consistent (i.e., has non-empty candidate set). In some sense, these are the only required steps. The refinement stops when the solution construction function (in `Find-Plan` procedure) can construct a solution candidate from the search node it picks for refinement.

An important consideration in refinement planning is cost of refinement. Complete consistency check turns out to be NP-hard for most common auxiliary constraints, making refinement cost non-polynomial. When 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. It is to this end that some planners use either a pre-ordering step (such as total ordering), or a conflict resolution step. The net effect of both these steps is to further refine the refinements generated by the establishment step, by adding additional ordering and binding constraints, until the consistency of each refinement can be checked with cheaper (polynomial time) consistency checks (see next section). In contrast to conflict resolution step, which is explicitly geared towards consistency check, pre-ordering is in general geared towards making handling of partial plan, including consistency check and truth criterion interpretation, tractable.

From the description, it is clear that refinement (or candidate set splitting) is done in three different stages-- as a part of establishment of new goals (we call this the *establishment refinement*), in pre-ordering the plan (called *pre-ordering refinement*), and in conflict resolution (called *conflict-resolution refinement*). In each refinement strategy, the added constraints include step addition, ordering addition, binding addition, as well as addition of auxiliary constraints. Although, for the sake of exposition the algorithm is written with the three refinement strategies in a particular serial order, they can be done in any particular order. Conceptually, we can imagine the planner's main search loop to consist of picking a partial plan from the search queue, picking one of the refinement strategies, and generating refinements of the partial plan with respect to that refinement, pruning inconsistent refinements. The planner will never have to backtrack on the choice of refinement strategies -- the completeness of the algorithm will be preserved irrespective of the order in which the individual refinements are employed.

As we shall discuss in the next section, the traditional instantiations of all three individual refinement strategies (see below) can be shown to be *complete* in that every ground operator

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

  1. Construct the null plan $\mathcal{P}_\emptyset : \langle \{t_I, t_G\}, \{t_I \prec t_G\}, \emptyset, \{t_I \rightarrow \texttt{start}, t_G \rightarrow \texttt{finish}\}, , \emptyset \rangle$, where $t_I$
     is a dummy action with no preconditions, and the assertions in $\mathcal{I}$ as its effects, and $t_G$ is a dummy
     action with no effects, and the assertions in $\mathcal{G}$ as the effects.

  2. Initialize search queue with $P_\emptyset$

  3. Begin Loop

     (a) Nondeterministically pick a node $\mathcal{N}$ with a corresponding partial plan $\mathcal{P}$ from the search
         queue.
     (b) If sol($\mathcal{P}$) returns a solution, return it, and terminate.
     (c) If sol($\mathcal{P}$) returns $*fail*$, skip to Step 3a.
     (d) Call Refine-plan($\mathcal{P}$) (See Figure 3) to generate refinements of $\mathcal{P}$.
     (e) Add all the refinements to the search queue; Go back to 3a.

     End Loop
```

Figure 2: A generic planning algorithm for refinement planning: Initial Call

sequence belonging to the candidate set of the parent plan will be a candidate of at least one of the refinements generated by each refinement strategy. Thus, all instantiations of `refine-plan` with these refinement strategies are also complete in the following technical sense: If there is a solution candidate in the candidate set of the initial (null plan), and if the solution constructor function is capable of picking that solution candidate given a sufficiently refined candidate set containing it, then any instantiation of `Refine-Plan`, with such a solution constructor function, will eventually terminate with success under all admissible search regimes.

.

4 A Rational Reconstruction of main ideas of Refinement Planning

In this section, we shall use the generic refinement planning algorithm formulated in the previous section as a backdrop to provide a rational reconstruction of some of the main ideas of refinement

Algorithm Refine-Plan($\mathcal{P}$)
Parameters: `pick-open`: the routine for picking open conditions. `pre-order`: the routine which adds orderings to the plan to make conflict resolution tractable. `conflict-resolve`:the routine which resolves conflicts with auxiliary constraints.

1. Goal Selection: Using the `pick-open` function, pick an open goal $\langle C, n \rangle$ (where C is a precondition of node n) from $\mathcal{P}$ to work on. *Not a backtrack point* (See Section 4.1.1)

2.1. Goal Establishment: Non-deterministically select an establisher step n' for $\langle C, n \rangle$. Introduce enough constraints into the plan such that (i) n' will make C true, and (ii) C will persist until n. n may either already be in the plan, or may be a new step introduced into the plan. *Backtrack point; all establishers need to be considered* (See Section 4.1)

2.2. Book Keeping: Add auxiliary constraints noting the establishment decisions, to ensure that these decisions are not violated by latter refinements. The auxiliary constraints may be one of goal protection, protection intervals or contributor protection. (see Section 4.2)

3. Refinements to make plan handling/consistency check tractable (Optional) This step further refines the refinements generated by establishment decision.

- **3.a. Pre-Ordering:** Use some static mechanism to impose additional orderings between steps of the refinements generated by the establishment check, with the objective of making handling of the partial plan (including consistency check, truth criterion interpretation) tractable *Backtrack point; all interaction orderings need to be considered.* (See Section 4.3.1)

  OR

- **3.b. Conflict Resolution:** Add orderings and bindings to resolve conflicts between the steps of the plan, and the plan's auxiliary constraints. *Backtrack point; all possible conflict resolution constraints need to considered.* (See Section 4.3.2)

5. Consistency Check: If the partial plan is consistent, return it. If it is inconsistent (i.e., has no safe ground linearizations), prune it. (See Section 4.3).

Figure 3: A generic algorithms for refinement planning: Refinement Procedure

planning algorithms. We will pay special attention to the search space characteristics. Unless specifically stated otherwise, for ease of exposition, we will be addressing the refinement of ground partial plans.

4.1 Establishment

The most fundamental refinement operation is the so-called establishment operation. This consists of adding constraints to the partial plan to establish some open condition $\langle C, n\rangle$ of the plan, where C is a precondition of step n in the partial plan of the search node. Establishments can be done with the help of existing steps in the plan (aka simple establishment), or by adding a new step to the plan (aka step addition). In either case, establishment introduces new subgoals (in the form of the preconditions and secondary preconditions of the steps taking part in establishment; see below), as well as new ordering and binding constraints. The process continues until all open goals are established.

First formalizations of refinement planning (e.g. [27, 1]) assumed that goal establishment requires ability to check the truth of arbitrary conditions in partial order plans. The Q&A procedure in Tate's Nonlin [27], provided the first formal specification of necessary and sufficient conditions for ensuring the truth of a proposition in a ground partially ordered plan. Later work by Chapman's TWEAK [1], extended this truth criterion to partially ordered partially instantiated plans. (see [1, 9] for more details). The truth criteria provide a way of checking the executability of a plan (and, for goals of attainment, also provide a way of checking goal satisfaction). When a particular condition is not true, the establishment step adds enough constraints to make it necessarily true by the truth criterion.

A problem with truth criterion based approaches is that interpreting truth-criterion becomes NP-hard for anything more expressive than TWEAK action representation.[9] This makes refinement non-polynomial. Since the objective of planning is to *make* a condition necessarily true, rather than *check* if a condition is necessarily true, it should be possible to avoid interpreting truth criterion, and just add sufficient constraints on the plan to ensure that the condition is necessarily true[10], thus delegating all constraint satisfaction to the consistency check step. Pednault's secondary preconditions based theory of establishment [20, 21] was the first to formally advocate this approach. In Pednault's theory, establishment essentially involves selecting some step n' (either existing or new) from the plan, and adding enough constraints to the plan such that (a) $n' \prec n$ and (b) n' causes C to be true after it (Pednault calls these causation preconditions $\sum_{n'}^{C}$) and (c) C is preserved by every step s' possibly coming in between n' and n (these are the preservation preconditions $\prod_{n}^{C}$). For Tweak action representation, causation preconditions are expressible as codesignation constraints among the variables of the plan, while preservation conditions are expressible as non-codesignation constraints. For actions with conditional effects, causation and

[9]In fact, even within TWEAK representation, allowing finite-domain variables will make truth-criterion NP-hard
[10]This essentially pushes the complexity into the consistency check.

preservation preconditions may also involve new subgoals at the corresponding steps. In that case, these subgoals are added to the list of open conditions that the planner has to work on.

4.1.1 Goal Selection

The establishment step assumes that the planner has selected a particular precondition of the plan for establishment in the current iteration. As far as the soundness and completeness of the planning algorithm is concerned, the goal selection can be done in any arbitrary fashion, and a refinement planner will never have to backtrack on the goal selection decision. There is a considerable amount of lee-way in selecting the goals. If precondition abstraction hierarchies are available, we can use them to guide the selection. Even when such information is not available, there may be several static goal ordering mechanisms. One particular static goal selection algorithm that is used in many planners (such as UA and TWEAK) is to pick a goal that is not necessarily true. The cost of this technique varies based on the nature of the partial plans maintained by the planner, and the expressiveness of the action representation used. For unrestricted partially ordered plans containing actions with conditional effects, this goal selection strategy is NP-hard. For planners that maintain restricted types of partial orderings, such as total orderings, or unambiguous orderings (c.f. [15]; also see Section 4.3.1), this cost can be tractable. It is important to note that although in the past different ‘‘brand-name’’ planning algorithms have been associated with different goal selection strategies, the decision on which selection strategy to pick is relatively independent of the way rest of the steps of the algorithm are instantiated. For example, there is no reason why SNLP cannot use an MTC based goal selection strategy.

4.2 Book Keeping and Posting of Auxiliary Constraints

4.2.1 Motivation

It is possible to limit refinement to establishment operations alone and still get a sound and complete planner. Chapman’s Tweak [1] is such a planner. It continues the establishment cycle until all the goals and preconditions are necessarily true. Unfortunately, TWEAK turns out to have a lot of redundancy in its search space. In particular, the example (a) in Figure 4 (originally from [15]) shows that TWEAK may find the same solution candidate in multiple different search branches. (That is, the candidate sets of the search nodes in different branches overlap.) Furthermore, since TWEAK does not keep track of which goals it already achieved and which it is yet to achieve, it may achieve and violate a goal/subgoal arbitrarily many times within the same refinement branch. This causes unnecessary looping and can generate many non-minimal solutions. Example (b) in Figure 4 shows a scenario where TWEAK repeatedly achieves and clobbers the conditions Q and R at action $a1$. In fact, in this example, TWEAK would loop infinitely if the action $a4$ is not available.

Although the presence of looping and non-minimal solutions in the search space is not so much of a problem for planners using best-first search strategies and addressing solvable problems (assuming that the ranking function takes into account the number of steps in the plan), it can be much more problematic when the underlying search strategy is depth-first. In Example (b) of Figure 4, a depth first search would have wasted a considerable amount effort on the looping branch, before reaching the depth bound and backtracking. This is where the book-keeping and conflict resolution steps comes in. Because the origins and motivations behind book keeping approaches are generally misunderstood, in the following subsections, we will first provide a rational reconstruction of the motivations behind the various approaches.

4.2.2 Book-keeping through Goal Protection

The looping behavior in the TWEAK example above could have been avoided had TWEAK kept track of the goals that it had already worked on, and avoided working on them again. In particular, suppose in a partial plan, the condition $\langle C, n\rangle$ which has already been considered by TWEAK once, is seen to be necessarily false. In this case TWEAK can prune the plan without loss of completeness (i.e., no minimal solutions will be lost). The rationale is that when $\langle C, n\rangle$ was first considered, the planner would have considered all possible ways of establishing C at n, and thus pruning a plan where an already established condition is necessarily violated, does not affect completeness in terms of minimal solutions.[11] This type of protection is known as *goal protection*. In terms of the refinement planning algorithm in Figure 3, goal protection adds an auxiliary constraint of the form $\langle C, n\rangle$ with the semantics that every candidate of the partial plan must have C true before n.

One way of implementing this book-keeping is to keep a list of `closed-goals` (i.e., goals which have already been considered by the establishment refinement step once), and whenever a new step is introduced into the plan, recheck to see if any of the closed-goals are possibly violated. If such ‘‘interactions’’ are discovered, then the planner could add additional orderings (constraints) to prune any unsafe ground linearizations (so that these linearizations will not be rechecked). If this process leads to an inconsistent set of orderings, then the plan does not have any safe ground linearizations, and thus can be pruned.

This technique for goal protection can be improved slightly-- rather than check for violation of every previously achieved goal on `closed-goals` list every time, we can keep track of the establishment dependencies imposed at the time each goal $\langle C, n\rangle$ is made true, as a causal link of the form $\{n'\} \xrightarrow{C} n$, where n' is the step that was originally used to establish C at n. When new steps are introduced into the plan, we first check to see if the new step possibly violates this causal dependency (i.e., if it possibly comes in between n' and n and deletes C). Only when this happens, do we have to use the full-blown truth criterion to check whether or not C is necessarily true at step n. This is exactly the strategy followed by Tate's Nonlin[12] Nonlin maintained the

[11]This essentially bounds the maximum depth of the search space to be proportional to the number of preconditions of the solution. Notice also that doing this does not avoid the looping problem completely. It is still possible to have long chains of operators of type n' and n'' such that n' establishes C'' for n'', but needs C' which can only be given by n''; see below.

[12]In this paper, we will only be concentrating on the goal establishment and conflict resolution aspects of Nonlin,

causal dependencies in a structure called GOST, and used them to reduce the number of calls to the costlier truth criterion (Q&A procedure).

An important, and often misunderstood ([14]) point to note about Nonlin's strategy of keeping causal links is that causal links were merely book-keeping devices for facilitating goal protection. From the planner's point of view, the contributors of a causal link do not play any role in the semantics of auxiliary constraints. In particular, suppose Nonlin finds that one of the causal links of the current plan, $\{n'\} \overset{C}{\rightarrow} n$, is violated, but the condition $\langle C, n \rangle$ itself is still true according to Q&A (presumably because of the effects of some other step n'' in the plan). In this case, Nonlin simply replaces $\{n'\} \overset{C}{\rightarrow} n$ with $\{n''\} \overset{C}{\rightarrow} n$ and continues. Thus the causal link $\{n'\} \overset{C}{\rightarrow} n$ does not preclude C from being violated in the interval $\langle n', n \rangle$. It merely implies that $\langle C, n \rangle$ has been worked on once and the planner will ensure that it is not violated within this branch.[13]

4.2.3 Goal Protection by Protection Intervals

Although Nonlin does not use causal links as protection intervals, there have however been other planners, such as Pedestal [12], UCPOP [23], which do interpret a causal link $\{n'\} \overset{C}{\rightarrow} n$ as a protection interval and consider any step s that can possibly intervene between n' and n and delete C as a threat (or conflict) that needs to be resolved by adding additional constraints to the partial plan. To understand this use of protection intervals, we have to recall that unlike TWEAK and Nonlin, many planners do not explicitly use the truth criterion to check for goal protection. A major reason for this is that interpretation of a necessary and sufficient truth criterion is intractable for actions more expressive than TWEAK action representation. Whenever a new step is added, checking to see if any of the previously achieved goals, or causal links, are violated will make refinement intractable.[14] Furthermore, since our ultimate goal is to *avoid* violating previously achieved goals (rather than merely checking if they are violated), we can avoid interpreting truth criterion, and add constraints on the plan directly to ensure that newly added step preserves all the previously achieved goals. In Pednault's terms, this involves adding preservation preconditions to ensure that the new step will avoid violating the previously achieved goals.

Making the new step explicitly preserve *all* previously achieved goals is over-conservative and can lead to loss of completeness. What we really want to do is to make the new step s preserve a condition C if and only if C is required at some step n such that s comes possibly before n, and possibly after n' (where n' is an establisher of C). This is exactly where protection intervals

and ignore such additional features as hierarchical task reductions.

[13]In some sense, this is as it should be. After all, since classical planning within STRIPS representation merely needs that certain conditions be true at certain steps (the so-called ''point protections''), from the point of view of soundness and completeness, there is no motivation for protecting a condition over a specific interval. (We are assuming that the planner is only interested in goals of attainment. The situation changes if we are also interested in goals of maintenance: e.g.: Hold block ''A'' in the right hand, while stacking blocks ''B'' and ''C'' on ''D'' and ''E'' respectively. Here, the protection intervals are necessary as a part of goal satisfaction.)

[14]Even Nonlin's method of keeping causal links and checking if the new step violates the causal link is still intractable for expressive action representations.

Figure 4: 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.

help. Specifically, we say that the new step s has to preserve a condition C if and only if there is a protection interval $\{n'\} \xrightarrow{C} n$ belonging to the plan such that s can possibly come between n' and n. In this case, either we can order s to not come in the interval $\langle n', n\rangle$, or let it come but add the preservation preconditions $\prod_s^C$ to the plan. This is the strategy followed by planners such as Pedestal [12] and UCPOP [23].[15] This strategy effectively shifts the complexity of consistency check from refinement to search space size.

4.2.4 Redundancy elimination through Contributor Protection

Although goal protection avoids working on the same goal more than once within the same branch, it does not completely remove redundancy from the search space. Specifically, the same candidate plan may be looked at in many different search branches. McAllester and Rosenblitt [14] were the first to point out that by using causal links as strong commitments to contributors, rather than as protection intervals or as support structures for goal protection, it is possible to make the planner systematic in the sense that no two search nodes in different branches will have overlapping candidate sets. (Note: a formal characterization of systematicity appears in Section 5.2.1). Specifically, McAllester suggests treating a causal link $\{s\} \xrightarrow{p} w$, as the commitment that s will be the effective contributor (i.e., last step before w that provides p) of p for w in every solution

[15]SNLP is functionally very similar, but it is best understood in terms of contributor protection.

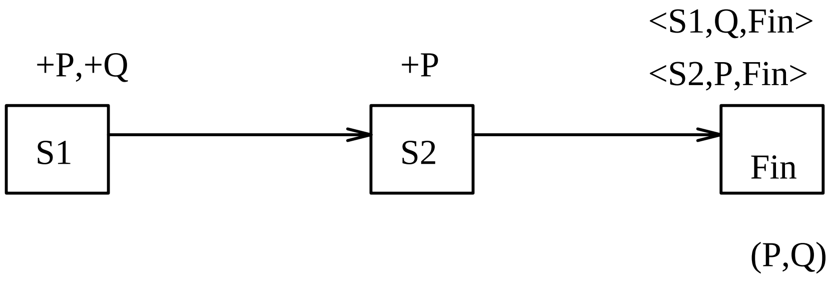

Figure 5: A non-minimal plan produced in spite of exhaustive causal links

derived from the partial plan. Causal links with these semantics are called **exhaustive causal links** (c.f. [6]).

Seen as an auxiliary constraint, an exhaustive causal link $\{s\} \xrightarrow{p} w$ has the semantics that no candidate of the partial plan shall have an operator adding or deleting p between the operators corresponding to s and w. It can be shown that posting exhaustive causal links as auxiliary constraints removes the redundancy in the search space by guaranteeing that no two partial plans in different branches of the search tree can have overlapping candidate sets (see Section 5.2.1).

Once again, it is easy to see that soundness and completeness are not affected by this form of commitment. The rationale is that at the time $\langle p, w\rangle$ was considered by the planner for expansion, all possible operators that can establish p at w have been considered and placed in different search branches. Thus pruning a branch where the original establisher of that branch no longer holds, does not lead to incompleteness (since the any eventual new establisher for $\langle p, w\rangle$ will also be present in one of the choices that were considered originally). It is also easy to see that that commitment to exhaustive causal links also implies that the planner will never work on any goal more than once.

It is instructive to note that although exhaustive causal links avoid the redundancy, and prevent looping of the kind discussed above, they are not completely immune to non-minimal solutions. For example, the plan shown in Figure 5 does have each precondition supported by a safe exhaustive causal link. However, solutions based on it are clearly non-minimal since S2 can be removed without violating the correctness of the plan. If S2 were to be a more general action with conditional effects, then all the work done on establishing the causation preconditions $\sum_{S2}^{P}$ is wasted, and any steps resulting from that will be redundant.

4.3 Consistency Check, Pre-ordering and Conflict Resolution

We have seen above that in terms of our refinement planning framework, all book keeping strategies add auxiliary constraints on the plan. The auxiliary constraints can all be represented in terms of causal links of the form $\{n'\} \xrightarrow{P} n$, where an effect of step n' was used to make P true at n. The semantics of the auxiliary constraint however depend on the particular book keeping strategy. For book keeping based on goal protection (e.g., NONLIN [27]), the auxiliary constraint requires that every candidate of the partial plan must have P true before the operator corresponding to step

n. For book keeping based on protection intervals (e.g., PEDESTAL [12]), the auxiliary constraint requires that no candidate of the partial plan can have P deleted between operators corresponding to n' and n. Finally, for book keeping based on contributor commitment, the auxiliary constraint requires that no candidate of the partial plan can have P either *added* or deleted between operators corresponding to n' and n.

The idea of consistency check is to prune inconsistent nodes (i.e., nodes with empty candidate sets) from the search space, there by improve the performance of the overall refinement search. (Thus, from completeness point of view, consistency check is really an optional step.) The consistency check involves ensuring that the partial plan has at least one safe ground linearization. This requires checking each ground linearization against all the auxiliary constraints. For general partial orderings, consistency check is intractable in the face of any of the above auxiliary constraints. This is where the pre-ordering and conflict resolution steps come in to play a useful role. Both of them aim to make the consistency check polynomial. In the case of pre-ordering, this aim is achieved by restricting the type of partial orderings in the plan such that we can check consistency with respect to auxiliary constraints without explicitly enumerating all the ground linearizations. In the case of conflict resolution, the partial plan is refined until each possible violation of the auxiliary constraint (called conflict) is individually resolved. In the process, additional orderings and bindings are added to the partial plan, and checking consistency will now amount to checking ordering and binding consistency. The former is always polynomial, while the latter is polynomial for variables with infinite domains.

4.3.1 Pre-ordering for tractable plan handling

The general idea behind pre-ordering strategies is to limit the class of partial plans encountered by the planner to some restricted class which can be handled in a tractable fashion.

From a consistency check point of view, pre-ordering strategies use some static mechanism to split the partial plan by introducing additional orderings into it such that the consistency check doesn't have to deal with exponential ground linearizations of the partial plan. One well-known pre-ordering approach is to introduce orderings into the partial plan until it becomes totally ordered. In this case, the consistency check can be done in polynomial time since there is only one linearization to worry about. A more interesting approach, proposed by Minton et. al. [16, 15], is to introduce orderings until the plan becomes *unambiguous*.

Definition 11 (Unambiguous Plan) *A ground partial plan $\mathcal{P}$ is said to be* **unambiguous** *, if every precondition $\langle C, n\rangle$ of the plan is either true in all completions (ground linearizations) or false in all completions.*

Unambiguity is achieved by ordering every pair of steps that can possibly interact with each other. Two steps t_1 and t_2 interact with each other if t_1 has an add list literal p and p is either a precondition or a delete literal of t_2.[16] For unambiguous plans, goal protection and protection interval constraints can be checked in polynomial time -- we simply have to pick one ground linearization and see if it is consistent. If it is, then every linearization is consistent; if it is not, then every linearization is inconsistent. It is also possible to get polynomial consistency check for exhaustive causal links by generalizing the definition of interaction to include the case where two steps share a common effect (see Section 4.5).

Apart from making consistency check tractable, pre-ordering strategies also help bound the cost of interperting modal truth criterion over pre-ordered plans. 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 goal selection and termination tests. In fact, unambiguous plans were originally used in UA [16] for this purpose.

4.3.2 Conflict Resolution

As discussed in the beginning of Section 4.3, conflict resolution step can be seen as a method to resolve all possible violations of auxiliary constraints one by one, so that the consistency of the resulting plan can be checked by a polynomial time order consistency check.

To do this, conflict resolution strategies keep track of potential violations of each auxiliary constraint. Suppose the auxiliary constraints in question are exhaustive causal links. An exhaustive causal link $\{s\} \xrightarrow{p} w$ $(s, w \in T)$ is possibly violated whenever there is a step t that can come between s and w in some linearizations, and either t adds p (in which case t rather than s is the effective contributor of p to w), or t deletes p (in which case, some step that comes necessarily after t must eventually be the effective contributor). This potential violation of $\{s\} \xrightarrow{p} w$ by t is considered a **conflict**, and t is considered a **threat** to the causal link. If t is adding p then it is called a called a **+ve threat** of the causal link, and if t is deleting p, then it is called a **-ve threat**.

Given a conflict consisting of a causal link $\{s\} \xrightarrow{p} w$ and a threat t, it is clear that no ground linearization of the partial plan where t can come between s and w will ever be consistent with respect to $\{s\} \xrightarrow{p} w$, and thus will not be a safe ground linearization. Equivalently, any ground linearization that is consistent with $\{s\} \xrightarrow{p} w$ will have t either before s or after w. Thus, resolution of this conflict consists of generating two refinements of the partial plan, one with the additional ordering $t \prec s$ and the other with the ordering $w \prec t$. At this point, the conflict between t and $\{s\} \xrightarrow{p} w$ is resolved in both these refinements. Conflict resolution continues by detecting further conflicts in these refinements, and resolving them by generating further refinements.

At the end of this process, enough ordering constraints[17] would have been added to the plan to resolve all conflicts. At this point, the consistency of the partial plan just depends on its order

[16]Note that in the description of UA in [16], Minton et. al. assume an action representation in which each delete list literal is included in the precondition list. Because of this, they define interaction solely in terms of overlap between effects of t_1 and preconditions of t_2. The restriction on the action representation is however an unnecessary one. Here, we remove the restriction, and change the interaction definition accordingly.

[17]for non-propositional plans, the constraints may also involve non-codesignation constrains

consistency, which is polynomial. It is instructive to note that a side effect of full conflict resolution is that every consistent plan is also safe, i.e., all its ground linearizations are safe (see Section 3.1). Similar threat and conflict definitions can be made for other auxiliary constraints. As remarked in Section 4.2.3, conflict resolution in more expressive action representation may involve adding additional sub-goals to the plan (as dictated by the preservation preconditions).

4.4 Solution Constructor function

As we discussed earlier, conceptually the only constraints on `sol` are that it signal success if it is able to find a solution candidate from the candidate set of the search node; and failure if it can guarantee that the candidate set does not contain any solution candidate. If it does neither, then the search node will be refined further. Broadly, there are two types of termination conditions that have been used by planners dealing with attainment goals:

MTC Based Solution Constructor (termination condition) Using the modal truth criterion, test if the preconditions as well as secondary preconditions (posted by establishment and conflict resolution steps) of each of the steps are necessarily true in the partial plan. If the test succeeds, then return an operator sequence matching a safe ground linearization of the plan.

Protection Based Solution Constructor (termination condition) Check if every precondition of every step in the given partial plan is supported by a causal link, and none of the preconditions are violated in the range specified by the supporting causal link (equivalently, no causal link has a -ve threat). If yes, then return an operator sequence matching a safe ground linearization of the plan.

The cost of the MTC based termination check depends on the expressiveness of action representation as well as the flexibility of plan representation. For the general class of partially ordered plans, it is polynomial only for TWEAK action representation, and becomes NP-hard if actions are allowed to have conditional effects. It is, however, possible to keep this termination check tractable if additional restrictions are imposed on the class of partial orderings. In particular, if the partial plans are either *totally ordered* or *unambiguously ordered*, then the termination check can be carried out in polynomial time (see Section 4.3.1).

The cost of protection based termination check depends largely on the particular conflict resolution strategy used by the planner. In planners such as Pedestal, SNLP and UCPOP, which incrementally keep track of conflicts for every causal link, and resolve them during planning, this check essentially involves looking up to see if there are any open conditions (a plan precondition that is not yet considered by the establishment refinement), and if there are any threats to the existing causal links. If the answer to these queries is no, then the termination check succeeds. This check can be done in $O(1)$ time.

Notice that in order to be able to use the protection-based termination check, the planner must establish each plan precondition individually. In contrast, the MTC based termination check may

Planner	**Termination Check**	**Goal Selection**	**Book-keeping**	**Strategy used to make consistency check tractable**
Tweak [1]	MTC-based ($O(n^4)$ for TWEAK rep; NP-hard with Cond. Eff)	Pick if not nec. true ($O(n^4)$ for TWEAK rep; NP-hard with Cond. Eff)	None	None
UA [16]	MTC-based $O(n^4)$ always	Pick if nec.false $O(n^4)$ always	None	Unambiguous ordering
Nonlin [27]	Q&A based	Arbitrary $O(1)$	Goal Protection via Q&A	Conflict Resolution (aided by Causal links)
TOCL [24]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Total ordering
Pedestal [12]	Protection based $O(1)$	Arbitrary $O(1)$	Goal Protection by protection intervals;	Total ordering
SNLP [14]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Conflict resolution
UA-SNLP (Section 4.5)	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 several existing planning algorithms in terms of the generic algorithm `Refine-Plan`

terminate even before all goals are explicitly established by the planner. The important point to note however is that the choice of termination test is to some extent independent of the way the rest of the algorithm is instantiated. For example, even though SNLP algorithm given in [14] used a protection-based termination test, it can be easily replaced by the MTC based termination check, with its attendant tradeoffs.

It is also important to remember that the two solution constructor functions discussed above are by no means the only alternatives. The only fundamental constraint on the solution constructor function is that it pick a solution candidate from the candidate set of the given partial plan (if one exists). There is complete flexibility on the way this is accomplished. For example, it is possible to formulate a constructor function that searches in the space of ground linearizations of the partial plan, looking for a safe ground linearization, and if one is found, returns the matching candidate. This latter check is clearly exponential for general partial orderings. But, it has the advantage of avoiding the dependency on the truth criteria or conflict resolution completely. The issue of whether or not such strategies will provide better overall performance than the two strategies discussed above, still needs to be investigated.

4.5 Integrating Competing Approaches: Case study of UA-SNLP

Table 1 characterizes a variety of planning algorithms in terms of our generic refinement planning algorithm. One of the important benefits of our unified framework for refinement planning algorithm is that it clarifies the motivations behind main ideas of many refinement planners, and thus makes it possible to integrate competing approaches. As a way of demonstrating this, we describe a hybrid planning algorithm called UA-SNLP, which uses book keeping based on exhaustive causal links (thus ensuring systematicity, see Section 5.2.1), but achieves tractable consistency check via a pre-ordering strategy motivated by that used in UA, rather than a conflict resolution strategy (see Table 1). In the following, we describe the pre-ordering strategy of UA-SNLP.

Given a partially ordered plan $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$, two steps t_1, t_2 are said to be interacting if they are unordered with respect to each other, t_1 has an effect p and.

1. either t_2 has p in its precondition list **or**
2. t_2 has p in its delete list **or**
3. t_2 has p in its add list

The pre-ordering step for UA-SNLP consists of ordering any newly introduced step with respect to each step with which it interacts. Once the step is pre-ordered this way, the consistency check with respect to the exhaustive causal links can be done in polynomial time, since either every linearization of the plan is safe, or no linearization is safe. Thus, for each partial plan coming out of the pre-ordering step, we can simply take a single linearization of that plan and check if that linearization satisfies the causal links. If it does, then the plan as a whole is included in the search queue. If any of the causal links are violated by the linearization, then the plan as a whole has no safe linearizations, and can be pruned.

It is worth noting that if we are only interested in goal protection based conflict resolution, then we can ensure polynomial time conflict resolution even without the third check in the definition of interaction. The first two are sufficient to guarantee unambiguity of partial plan, which in turn would guarantee that each precondition is either necessarily true or necessarily false. The last case adds additional ordering to ensure that +ve threat resolution is also polynomial. The addition of this last case provides a sufficient (rather than necessary) grounds for ensuring systematicity of the overall algorithm. The polynomial conflict resolution can be retained for more expressive action representations containing conditional effects by making the definition of interaction more conservative (as described in [16]).

UA-SNLP presents an interesting contrast to SNLP -- both are systematic and provide tractable consistency check. While SNLP achieves tractable consistency check by conflict resolution, UA achieves it by a pre-ordering strategy that depends on interactions between steps. Since UA-SNLP's notion of interaction between steps does not depend on the causal links supported by the respective steps, UA-SNLP will order more than SNLP will. On the flip-side, however, UA-SNLP avoids the necessity to detect conflicts, and more importantly, ensures tractable truth criterion check. This latter ability allows UA-SNLP to use goal selection and termination criteria that are intractable for SNLP (for more expressive action representations).

We have implemented UA-SNLP, and have found that in some circumstances UA-SNLP does better than UA as well as SNLP. A more comprehensive evaluation of this algorithm, as well as a generalization implemented over UCPOP, a causal link based planner for actions with conditional and universally quantified effects, are currently under way. A further discussion of some of the properties of UA-SNLP algorithm appears in Section 5.5.

5 Search Space size, refinement cost and Performance

In this section, we will demonstrate the analytical power of the unified framework, developed in the previous sections, by using it as a backdrop to analyze a variety of search space size vs. refinement cost vs. performance tradeoffs in refinement planning. We will start by developing a model for estimating the search space size of a refinement planner, and use it to characterize and analyze the tradeoffs of the various approaches to bounding search space size. Throughout this section, we will see that there is an ever-present tradeoff between refinement cost and search-space size: most approaches for reducing search space size increase the refinement cost. Because of this, polynomial refinement cost will be an important consideration in evaluating various approaches for reducing search space size. Unless otherwise stated, we will be considering refinement of ground partial plans for ease of exposition.

5.1 A model for estimating search space size

Let $\Im$ be the set of all candidates (ground operator sequences) for a refinement planner. To characterize the size of the refinement search space, we shall look at a *termination search tree*, which is any search tree such that all the leaf nodes of the tree are either solution nodes (i.e., `sol` will return a candidate) or failure nodes (i.e., either it has no further refinements, or `sol` will return *fail*, implying that there is no solution candidate in the candidate set of that node). We shall call the set of leaf nodes of the termination tree, the *termination fringe*, $\wp$ of the refinement search. In the following, we shall be using the size of the termination fringe as a measure of the size of the search tree.[18]

At the outset, we recognize that union of candidate sets (denotations) of all the nodes in the termination fringe must be equal to the overall candidate space ($\Im$). To get an idea of the size of the termination fringe, we need to have an idea of the average size of the denotations of the fringe nodes. Let us denote this quantity by $\xi \geq 0$. Since the same candidate may fall in the candidate set of more than one node, we also need to take into account the measure of redundancy. Let $\rho(\geq 1)$ be the average number of fringe nodes whose denotations contain a given candidate. With this information, we can estimate the size of the termination fringe as:

$$|\wp| \times \xi = |\Im| \times \rho \tag{1}$$

$$|\wp| = \frac{|\Im| \times \rho}{\xi} \tag{2}$$

[18]This is reasonable since the total size of the search tree is of the order of the size of its fringe.

Search space size is only one factor affecting the worst case performance of a planner. The other factor is the average cost of refinement. If $\Re$ is the average cost of refinement, then the worst case planning effort is given by the product of search space size and refinement cost:

$$Cost = \Re \times |\wp| \tag{3}$$

From equation 2, it is clear that the larger ξ is and the closer to one ρ is, the smaller the size of the search space. Reducing ρ involves designing refinement operators that split the candidate set of a node into non-overlapping subsets. This idea is formalized as systematicity below. The size of ξ is affected by several factors.

Least Commitment ξ will be larger if the refinement operators, on the average, split a node's candidate set into fewer number of children nodes. In the case of planning, this means that all else being equal, planners that do not employ conflict-resolution and pre-ordering step (in the algorithm in Figure 3) will have larger ξ than planners which use them.[19] However, the latter will in general have non-polynomial refinement cost, since consistency check without conflict resolution or pre-ordering is NP-hard in general.

Stronger termination check The earlier in search we are able to pick a solution candidate, the larger is the value of ξ. In the case of planning, a planner that uses a termination check based on TWEAK truth criterion will be able to terminate before explicitly establishing each open condition. It thus leads to larger ξ values than one which uses a protection based termination condition, requiring the planner to explicitly work on each of the goals. Thus, the former has a larger value of ξ (and thus smaller search space) than the latter, all else being equal.

Stronger pruning check ξ will be larger if the refinement search is able to prune unpromising nodes earlier. In the case of planning, this means that a planner with a `sol` function that has a strong pruning component will have a smaller search space than one without such a pruning component.

We will discuss the utility of each of these approaches to reducing search space size in turn. These discussions will foreground the fact that there is a tradeoff between refinement cost and search space size: most methods for reducing $|\wp|$ wind up increasing $\Re$.

[19]It is important to note that search space size is determined by the combined effect of ξ and ρ. For example, in [16], it is shown that even though TWEAK avoids pre-ordering step used by UA, TWEAK can nevertheless have a larger search space size. This is because of redundancy in TWEAK's search space: the larger value of ξ is offset by a correspondingly larger value of ρ in TWEAK.

5.2 Bounding Search Space with the size of the candidate Space: Systematicity and Strong Systematicity

In this section, we will look at the notions of ''systematicity'' in planning from the point of view of refinement search. We will clarify the bounds systematicity places on the size of the search space, and look at the costs and benefits of enforcing those bounds.

5.2.1 Systematicity

In terms of bounding the search space size, the minimal guarantee one would like to provide is that the size of the search space will not be more than the size of the overall candidate space $|\Im|$. Trying to do this gets us to two important notions of irredundancy in refinement planning: systematicity and strong systematicity.

Definition 12 (Systematicity of Refinement Search) *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, the candidate sets represented by $\mathcal{N}$ and $\mathcal{N}'$ are disjoint. That is,* $[[\mathcal{N}]] \cap [[\mathcal{N}']] = \emptyset$.

Since the solution constructor function of a refinement search, by definition, looks for a solution candidate within the candidate set of a search node, if the candidate sets of search nodes on the termination fringe are non-overlapping (as is guaranteed by systematicity property), the search will never return the same solution candidate more than once. Thus, we have:[20]

Proposition 3 *A systematic refinement search will never return the same solution candidate in more than one search branch.*

Clearly, for a systematic planner, ρ, the redundancy factor is 1. The definition also implies that each (minimal) solution candidate is found in at most one search branch. The following property is a direct consequence of this definition.

Proposition 4 *In a systematic refinement search, the sum of the cardinalities of the candidate sets of the termination fringe will be no larger than the size of potential candidate set $\Im$. (I.e., if $\sum_{\mathcal{N}_i \in \wp} |[[\mathcal{N}_i]]| \leq |\Im|$).*

The straightforward, if highly impractical, way of guaranteeing systematicity would be to enumerate the candidate sets of every new search node, and remove the redundant candidates. A more practical option involves ensuring that individual refinements are systematic:

[20]Other researchers (e.g. [4]), have used this latter notion of solution systematicity as the definition of systematicity. This definition turns out to be problematic since for unsolvable problems, there are no potential solutions and thus any search strategy would be considered systematic by it. Our formulation of systematicity avoids this problem. It is also more general and entails their definition as a corollary. Finally, it also allows the formulation of the notion of strong systematicity (see below), and thereby providing an additional finer distinction between systematic planners.

Definition 13 (Systematic Refinements) *A refinement strategy $\mathcal{R}$ is said to be systematic if for every pair of refinements $\mathcal{N}_i, \mathcal{N}_j$ of a node $\mathcal{N}$ generated by $\mathcal{R}$, $[[\mathcal{N}_i]] \cap [[\mathcal{N}_j]] = \emptyset$.*

If every refinement strategy used in refinement search is systematic, then it is easy to see that the overall search is systematic. From our discussion in Section 4.1, we recall that the fundamental refinement operation in the `refine-plan` algorithm in Figure 3 is the establishment refinement (i.e., refining a partial plan to make an additional subgoal true). The other (optional) refinement operations are pre-ordering and conflict resolution.

We will start with the systematicity of establishment refinement. As we saw, book-keeping step is used to avoid undoing and/or repeating previous establishment decisions. establishment refinement adds auxiliary constraints. For the establishment refinement to be systematic, the auxiliary constraints must ensure that the candidate sets of the refinements will not overlap. McAllester and Rosenblitt [14] noted that adding exhaustive causal links as auxiliary constraints provides sufficient constraint on the candidate sets of the nodes to ensure non-overlapping candidate sets.

In terms of the algorithm in Figure 3, whenever a plan $\mathcal{P}$ is refined by establishing a condition $\langle C, n\rangle$ with the help of the effects of a step n', the exhaustive causal link $\{n'\} \xrightarrow{C} n$ is added to the refinement as an auxiliary constraint. Different refinements of $\mathcal{P}$ generated on the basis of the subgoal $\langle C, n\rangle$ will have different exhaustive causal links supporting $\langle C, n\rangle$. With this restriction, it can be shown that establishment refinement operation always splits the candidate set of a node into non-overlapping subsets.

As McAllester points out in [14], this property follows from the fact that exhaustive causal links provide a way of uniquely naming steps independent of step names. To understand this, consider the following partial plan:

$$\mathcal{N} : \langle \{t_I, t_1, t_G\}, \{t_I \prec t_1, t_1 \prec t_G\}, \emptyset, \{t_1 \rightarrow o_1, t_I \rightarrow \texttt{start}, t_G \rightarrow \texttt{finish}\}, \{\{t_1\} \xrightarrow{P} t_G\}\rangle$$

where the step t_1 is giving condition P to t_G, 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 literal Q. The establishement refinement generates two partial plans:

$$\mathcal{N}_1 : \left\langle \begin{array}{l} \{t_I, t_1, t_2, t_G\}, \{t_I \prec t_2, t_2 \prec t_1, t_1 \prec t_G\}, \emptyset, \\ \{t_1 \rightarrow o_1, t_2 \rightarrow o_2, t_I \rightarrow \texttt{start}, t_G \rightarrow \texttt{finish}\}, \{\{t_1\} \xrightarrow{P} t_G, \{t_2\} \xrightarrow{Q} t_G\} \end{array} \right\rangle$$

$$\mathcal{N}_2 : \left\langle \begin{array}{l} \{t_I, t_1, t_2, t_G\}, \{t_I \prec t_2, t_2 \prec t_1, t_1 \prec t_G\}, \emptyset, \\ \{t_1 \rightarrow o_1, t_3 \rightarrow o_3, t_I \rightarrow \texttt{start}, t_G \rightarrow \texttt{finish}\}, \{\{t_1\} \xrightarrow{P} t_G, \{t_2\} \xrightarrow{Q} t_G\} \end{array} \right\rangle$$

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 exhaustive causal links 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. Coming to the pre-ordering and conflict-resolution refinement strategies, both of these generate refinements by splitting the ground linearizations of the plan into non-overlapping subsets by introducing additional orderings between the steps of the partial plan. Since, as seen above, plan steps are uniquely identifiable in terms of causal relations when exhaustive causal links are used, these refinements also implicitly split the candidate set of the parent node into non-overlapping subsets, and are thus systematic. For example, consider a partial plan

$$\mathcal{N} : \left\langle \begin{array}{l} \{t_I, t_1, t_2, t_G\}, \{t_I \prec t_1, t_I \prec t_2, t_2 \prec t_G, t_1 \prec t_G\}, \emptyset, \\ \{t_1 \rightarrow o_1, t_2 \rightarrow o_2, t_I \rightarrow \texttt{start}, t_G \rightarrow \texttt{finish}\}, \{\{t_1\} \xrightarrow{P} t_G, \{t_2\} \xrightarrow{Q} t_G\} \end{array} \right\rangle$$

The pre-ordering and conflict resolution strategies might refine $\mathcal{N}$ by ordering the unordered steps t_1 and t_2 and both possible ways:

$$\mathcal{N}_1 = \mathcal{N} + (t_1 \prec t_2)$$

$$\mathcal{N}_2 = \mathcal{N} + (t_2 \prec t_1)$$

Thus in every candidate of $\mathcal{N}_1$, the last operator in the sequence giving effect P will always precede the last operator in the sequence giving effect Q. The opposite holds for every candidate of $\mathcal{N}_2$. Since the operators are uniquely identifiable in terms of exhaustive causal links, these constraints are sufficient to ensure that no ground operator sequence can be a candidate of both $\mathcal{N}_1$ and $\mathcal{N}_2$.[21] In other words, pre-ordering and conflict resolution refinements are also systematic in the presence of exhaustive causal links.

Thus, as long as the book-keeping step posts exhaustive causal links as auxiliary constraints, all the refinements used in the algorithm `Refine-Plan` in Figure 3 are individually systematic. Thus, any instantiation of `refine-plan` algorithm with exhaustive causal links, including UA-SNLP discussed in Section 4.5, is guaranteed to be systematic.

[21]Alternately, we can also show that in the case of plans with exhaustive causal links, every candidate of the plan is derivable from exactly one (safe) ground linearization (either as an exact match of the linearization, or as an augmentation of it). Thus, any refinement strategy that splits the ground linearizations of a plan resulting from establishment refinement into non-overlapping sets will also split the candidate set of the plan into non-overlapping subsets, thus preserving systematicity. The pre-ordering and conflict resolution strategies discussed in Section 4 all have this property, thus ensuring the systematicity of the overall algorithm.

Systematicity and overlapping linearizations: In the past, there has been a tendency to identify the redundancy in the search space with overlapping linearizations of partial plans in the search space (e.g. [15, 7, 4]). From the formulation in this section, we note that systematicity implies only that no two partial plans in different branches of the search can share a *safe* ground linearization. Sharing of unsafe ground linearizations doesn't affect systematicity since candidate set of a partial plan is related only to the safe ground linearizations (see Section 3.1). In particular, it is easy to see that no augmentation of an operator sequence matching an unsafe ground linearization will ever be a candidate for a partial plan with auxiliary constraints specified by exhaustive causal links. The proposition below thus follows:

Proposition 5 *Systematicity implies non-overlapping ground linearizations only when all the plans in the search space are* **safe** *(i.e., all their ground linearizations are safe).*

As remarked in Section 4.3.2, planners that do full conflict resolution (i.e., resolve all conflicts to auxiliary constraints before the next establishment cycle), maintain safe partial plans in their search space. If such planners are systematic, then no two plans in different branches in their search space will have overlapping ground linearizations.

Conflicting interpretations of Systematicity: There have been several conflicting interpretations of ''systematicity'' of partial-order planning in the literature. Some researchers (c.f. [3]) have interpreted McAllester's systematicity claim to mean that even though the same operator sequence may appear in more than one branch, it will never have the same causal structure. This notion of **causal structure systematicity** is considerably weaker than the notion of systematicity which we defined above, and which is guaranteed by SNLP. Systematicity is stated in terms of operator sequences rather than in terms of step names and causal structures. This is reasonable because step names and causal links are artifacts of the particular planning strategy, while systematicity should be defined with respect to the space of potential candidates.

5.2.2 Strong Systematicity

Given a systematic refinement planner we have:

$$|\wp| = \frac{|\Im|}{\xi}$$

Thus, to bound search space size by the size of candidate space, we only need to bound the average size of the candidate set of the fringe nodes, ξ. This however turns out to be tricky. In particular, it is possible for the nodes in the termination fringe to have an empty candidate set (i.e., there is no ground operator sequence satisfying the constraints of the partial plan corresponding to that search node). The presence of such nodes can drive down the average size of the candidate sets ξ, and thus increase the size of the search space (we shall see an example of this in Figure 6). One way to guarantee that $\xi \geq 1$ would be to ensure that there are no inconsistent nodes in the search space with empty candidate sets. This leads us to the notion of strong systematicity.

Definition 14 (Strong Systematicity) *A refinement search is said to be* **strongly systematic***, if and only if it is systematic and every node in the search tree is consistent.*

From the definition of strong systematicity, we have $\xi \geq 1$ and thus $|\wp| \leq |\Im|$.

Proposition 6 *The size of the termination fringe of a search tree generated by a strongly systematic refinement search is strictly bounded by the size of the candidate space (i.e. $|\Im|$).*

To convert a systematic search into a strongly systematic one, we only need to ensure that every node in the search space is consistent. Thus strong systematicity is guaranteed as long as the consistency check is strong enough to prune all inconsistent partial plans generated by the plan refinements. Here, once again, we see a tradeoff between search space size and cost of refinement: As we discussed in Section 4.3, checking consistency of general partial orderings against a set of exhaustive causal links is NP-hard [25]. The alternative of using pre-ordering or conflict resolution strategies to make consistency check tractable involves considerable amount of additional refinement over and above that required for establishment, and thus reduces ξ, there by increasing the search space size.

It is interesting to note that pre-ordering and conflict resolution strategies provide strong systematicity by ensuring that each partial plan in the search queue is *safe*. Strong systematicity, on the other hand, only requires *consistency* rather than safety. In Section 5.3, we will look at the possibility of delaying or ignoring resolution of some conflicts (there by reducing the search space size), while still preserving strong systematicity.

5.2.3 On the Utility of Systematicity

Having spent considerable effort in clarifying the nature of systematicity in refinement planning, we shall now explore the practical utility of systematicity. As our discussion in Section 4.2 demonstrated, systematicity is a natural culmination of efforts to reduce looping, redundancy and non-minimal solutions in search space of refinement planners. The fact that contributor protection, which in some sense is a natural extension of protection intervals, and goal protection, can provably eliminate redundancy from search space is quite pleasing.

In terms of the overall search space size, book keeping strategies, including those guaranteeing systematicity, essentially allow planner to stop refining a partial plan earlier than would be the case without book keeping (by adding auxiliary constraints to make the plan inconsistent). This tends to increase ξ and thereby reduce the search space size. At the same time, if consistency check is done via conflict resolution/pre-ordering to ensure tractability, then this introduces more refinement than would be done by a planner without any book-keeping strategy, and thus tends to reduce ξ and increase search space size. (*Note that the increase in the search space size in the latter case is not caused by systematicity, but rather by the way in which consistency check is implemented*)

Even if the search space size is smaller with systematicity, the performance of a planner does not depend solely on the worst case search space size. Like other forms of systematic search (c.f [11]), systematicity (or elimination of redundancy) in refinement planning is achieved at the expense of increased commitment -- in this case to particular establishment structures (or contributors). The elimination of redundancy afforded by systematicity (strong systematicity, to be precise) can be very helpful when the solution density is very low, as it prevents the planner from considering the same failing candidates more than once. However, when the solution density is not particularly low, redundancy in the search space is only one of the many factors affecting the performance of a refinement planner. Another (perhaps equally) important factor, in such situations, is the level of commitment in the planner. The use of protection intervals and causal links, while reducing redundancy, also result in increased commitment to particular establishment structures. Increased commitment leads to higher backtracking, which in turn can adversely affect the performance. This is in contrast to planners such as TWEAK, which avoid all forms of goal protection and contributor protection, but at the expense of increased redundancy.

In Section 4.2, we emphasized that systematicity should not be seen in isolation, but rather as part of a spectrum of methods for reducing redundancy in the search space. Our discussion in 4.2 suggests that every thing else (e.g., pre-ordering, conflict-resolution, termination and goal selection strategies) being equal, a planner that uses book-keeping based on exhaustive causal links (like SNLP), and a planner that ignores book-keeping (like TWEAK), are in some sense at the extremes of the spectrum of approaches to redundancy-commitment tradeoff in refinement planning. In [6, 7], we provided empirical support to the hypothesis that better approaches to this tradeoff may lie in the middle of these two extremes. In particular, we showed that planners using multi-contributor causal structures may out-perform both TWEAK and SNLP under certain circumstances. The latter maintain causal links, but do allow change of causal contributor during planning.

On a related note, the discussion in this paper also sheds some light on several extant misconceptions about the role and utility of systematicity property.[22] Since McAllester's SNLP was the first planner to state and claim this property, there has been a general tendency to attribute any and every perceived disadvantage of SNLP algorithm to its systematicity. For example, it has been claimed (e.g. [10]) that use of causal links and systematicity *increases* the effective depth of the solution both because it works on +ve as well as -ve threats, and because it forces the planner to work on each precondition of the plan individually. 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 exhaustive causal links in the book-keeping step, has a direct bearing on the systematicity of the algorithm. As we discussed above, and in Section 4.3, 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 conflict resolution and/or preordering strategies used in order to get by with tractable termination

[22]We have already addressed the misconceptions about the definition of systematicity property in the previous sections.

and consistency checks. As mentioned in Section 4.4, it is in principle possible to design an instantiation of `Refine-Plan` which only uses an establishment based refinement, coupled with a more flexible, if costlier, solution constructor function (such as MTC-based constructor). Such a planner clearly does not affect the solution depth -- both because it does not force the planner to work on each individual precondition separately, and because without conflict resolution, there is no notion of +ve (or -ve) threat resolution.

5.3 Reducing Search Space size by deferring conflicts

We have noted earlier (Section 4.3.2) that conflict resolution provides polynomial consistency check by further refining the refinements generated by the establishment step. From Equation 2, we can see that every time we split a search node, we reduce ξ and thus increase the search space size. Furthermore, many of the conflicts in planning are ephemeral, in that in the course of other goal establishments, they may naturally disappear. Thus, resolving conflicts as soon as they appear may split the candidate set of a search node more than necessary, thereby reducing ξ and increasing the search space size (see equation 2). This motivates the idea of deferring resolution of some conflicts. In this section, we shall evaluate a variety of ideas for conflict deferment in terms of our model of refinement planning.

To begin with, it is worth reiterating the fact that the strategies used to implement conflict resolution deferment and consistency check have no effect on the systematicity. Systematicity is a property of candidate sets, and as soon as the appropriate auxiliary constraint is added to a partial plan, its candidate set is determined. Further refinements of that partial plan by pre-ordering and conflict-resolution strategies, will not introduce any redundancy as long as they partition the candidate sets; as discussed in Section 5.2.1). More practically, since solution-constructor function will only check for solution candidate within the candidate set of the given plan, the planner will never return the same solution more than once.[23]

Strong systematicity, however, is a different matter. In general, arbitrary conflict deferment will preserve strong systematicity only when the consistency check employed by the planner is *complete*. As mentioned in Section 4.3, a complete consistency check is exponential in most cases. Many implemented planners, such as SNLP, UCPOP and PEDESTAL, use polynomial consistency check that is only complete in the presence of undeferred conflict resolution. Specifically, a partial plan may be inconsistent as a whole (i.e., has empty candidate set), but the deferment of conflict resolution makes polynomial consistency check miss this inconsistency (as it may not yet have translated into order/binding inconsistency). While the inconsistency will be detected ultimately, arbitrary amount of refinement work may be done on inconsistent nodes in the mean time; thus destroying strong systematicity. This point is illustrated in a telling fashion by the example in Figure 6, which compares complete search space generated by systematic SNLP with that generated by a variant called SNLP$'$ that postpones +ve threats, for the same planning problem.

[23]Although Harvey et. al. also make this point [4], our formulation of systematicity in Section 5.2.1 makes it self evident.

Figure 6: Example demonstrating that arbitrary threat deferment does not ensure minimal worst case search space size

The planner on the left is strongly systematic, while the one on the later is systematic but not strongly systematic. Clearly, SNLP$'$ has a larger search space than that of SNLP. More over, the additional nodes are all those with empty candidate sets (and will be pruned ultimately when the planner attempts to resolve the +ve threats). This also points out that all else being equal a planner which is merely systematic (and not strongly systematic) may have a *larger* search space than a corresponding planner that is not systematic. For example, it is easy to show that a planner which completely ignores +ve threats (such as Pedestal [12]), is un-systematic (i.e., contains overlapping candidate sets). However, it would stop before resolving the positive threats, and thus would generate a smaller search space than SNLP$'$. (Notice also that the search space of SNLP$'$ contains partial plans with overlapping linearizations. As remarked earlier, this does affect systematicity, since all the overlapping linearizations correspond to unsafe ground linearizations.)

In general, the only form of conflicts that can be deferred without affecting strong systematicity are those which can be provably resolved independently of how other conflicts are resolved (this ensures that consistency is not affected bya the deferment). However, it is often intractable to check dynamically whether an conflict is of this type; thus increasing the refinement cost. Smith and Peot [25] propose an attractive alternative that relies on a tractable pre-processing step to recognize *some* of the conflicts that are provably resolvable. Our discussion above shows that deferring conflicts based on such an analysis clearly preserves strong systematicity. In the following we will look at some static threat deferment strategies.

5.3.1 Static conflict deferment strategies

Our discussions on search space size have until now concentrated on propositional planning with TWEAK representation. When we go to non-propositional planning, an important question is regarding the definition of conflicts. Suppose we have a causal link $\{n'\} \overset{P(x,y)}{\rightarrow} n$ and a step s which can come between n' and n and which has a delete literal $P(u, v)$. The question is whether or not s should be treated as a conflict (threat) by SNLP. McAllester [14] avoids this problem by using an independent *lifting transformation* -- the question of whether $P(x, y)$ is equal to $P(u, v)$ is left to the lifting transformer, which will nondeterministically return either ''yes'' with binding constraints $[x \approx u, y \approx v]$ or ''no'' with one of the bindings $[x \not\approx u, y \approx v]$, $[x \not\approx u, y \not\approx v]$ and $[x \approx u, y \not\approx v]$. The planner simply adds these constraints to the binding constraints of the plan, and does conflict resolution (i.e., add either promotion or demotion) if the answer was yes; or ignores the conflict if the answer is no. If the added binding constraints are inconsistent with the existing binding constraints in the plan, the plan eventually gets pruned by consistency checking routines.[24]

We can in fact improve the search space size by avoiding calling the lifting transformer all together. We can say that s is a threat if and only if it can fall between n' and n and $P(u, v)$ *necessarily* codesignates with $P(x, y)$ modulo the binding constraints on the plan. Since most conflicts of this type of are ephemeral, which will disappear when the variables concerned get bound to constants in the process of establishing various goals, the strategy of ignoring a conflict until it is necessarily codesignating, avoids splitting the search nodes unnecessarily and thus increases ξ, (thereby reducing the size of the search space). Furthermore, as long as all variables occurring in the effects of an operator also occur in at least one of the preconditions (as is the case in most planning domains), and the initial state of the problem is completely ground, by the time all establishments are completed, every possible conflict involving a variable will either become a necessary conflict or a necessary non-conflict. Thus, there will never be a situation when the refinement cycle ends before resolving all conflicts.

Unfortunately however, this idea also can increase refinement cost for action representations more expressive than that of TWEAK. In particular, if we have variables with finite domains, checking whether the effects of a threat necessarily codesignate with a supported condition is NP-hard, thus making conflict detection itself intractable.

Further, deferring threats in this fashion does not in general preserve strong systematicity. If the variables have finite domains, it is possible that a set of conflicts are all individually separable, but all of them are not separable together. As an example, consider a partial plan with two conflicts:

[24]It is interesting to note that the lifting idea introduced in McAllester's paper, is not used in other implementations of SNLP. The most popular implementation of SNLP, done by Barrett and Weld, (hence forth referred to as UW-SNLP) defines s to be a threat to link $\{n'\} \overset{P(x,y)}{\rightarrow} n$ as long as $P(u, v)$ can possibly unify $\{s\} \xrightarrow{p} w$, if v has an add or delete literal q such that q can *possibly* unify with p modulo the bindings on the current plan. To resolve a threat, UW-SNLP either (a) adds the binding $[x \approx u, y \approx v]$ and promotes or demotes s or (b) orders s to come necessarily between n' and n (this avoids redundancy with respect to promotion and demotion strategies) makes one partial plan for each of the non-overlapping binding sets $[x \not\approx u, y \approx v]$, $[x \not\approx u, y \not\approx v]$ and $[x \approx u, y \approx v]$.

$\{t_1\} \overset{P(A)}{\rightarrow} t_2 \otimes t_3$ and $\{t'_1\} \overset{Q(B)}{\rightarrow} t'_2 \otimes t_3$, where t_3 has two delete list literals $P(x)$ and $Q(x)$, and x has the domain $\{A, B\}$. Here, both conflicts are individually separable, but in conjunction, they are unseparable. The only general way to guarantee strong systematicity in such cases is to check if the conflicts are resolvable *together*. The latter makes the deferment check intractable. This type of scenario does not occur for TWEAK representation which assumes variables with infinite domains, because for infinite domain variables, a set of non-codesignation constraints will never entail a codesignation constraint.

(In [22] Peot and Smith describe a planner called DSep which is a variant of SNLP that defers threats until they are ‘‘unseparable’’, i.e., necessarily codesignating, and provide a proof that DSep search space is no larger than that of SNLP which folds conflict resolution into establishment. Our discussion above shows that their proof holds only for the case of variables with infinite domains. For variables with finite domains, DSep will allow inconsistent nodes into its search space as well as refine them. SNLP which folds conflict resolution into establishment would have pruned such plans early on. Thus, it is possible for DSep to have a larger search space.)

5.4 Effect of Termination and Pruning Checks on Search Space Size

The solution constructor function, `sol`, used by the planner can also have an effect on the search space size of the refinement planner: The earlier in search we are able to pick a solution candidate, the larger the value of ξ. In the case of planning, a planner that uses a termination check based on a necessary and sufficient truth criterion will be able to terminate before explicitly establishing each open condition, than one which uses a protection interval/causal link based termination condition, requiring the planner to explicitly work on each of the goals. Thus, the former has a larger value of ξ (and thus smaller search space) than the latter, all else being equal. Once again, the flip side is that for general partial order planners with expressive action representations, checking necessary truth of a condition is intractable.

Another way the `sol` function can reduce the search space size is by pruning unpromising nodes earlier. Recall that `sol` function prunes a partial plan, when it can guarantee that there is no solution candidate in the candidate set of that node. By pruning such nodes as early as possible, the planner can increase the size of candidate sets of the failing nodes in the termination fringe, there by increasing ξ and reducing the size of the search space. Pruning involves computing macro characteristics of the candidate set of the node, analyzing it with respect to the outstanding goals, and showing that it does not contain a solution candidate with respect to them. This latter is best seen as a look-ahead computation. Few classical refinement planners employ this type of pruning. An example is the FORBIN planner [2], which does a significant amount of look-ahead scheduling before allowing a partial plan to enter the search queue and expanded. Once again, there is a tradeoff between the cost of the pruning strategy and the extent to which it can reduce the search space size [8].

5.5 Effect of pre-ordering and Conflict Resolution Strategies under Depth-first regimes

In the previous sections, we noted that pre-ordering and conflict resolution strategies, while providing tractable consistency check, also further refine the refinements provided by the establishment step. They thus increase the search space size by reducing ξ. While this would imply that pre-ordering and conflict resolution strategies would explore more nodes in breadth-first search regimes, it does not predict the size of the search space explored under depth-first search regimes. In this section, we will use our model of refinement planning to analyze these strategies under depth-first regime. To simplify the analysis, we will consider four planners that use contributor protection and guarantee strong systematicity: SNLP which uses conflict resolution to ensure tractable refinement; SNLP*, which is like SNLP, but uses a non-polynomial consistency check instead of conflict resolution; UA-SNLP which uses pre-ordering to ensure tractable refinement; and finally, TOCL which uses total ordering.

Since pre-ordering and conflict resolution split a search node into multiple search nodes, it is clear that the value of ξ is largest in SNLP*, and correspondingly smaller in SNLP, UA-SNLP, and TOCL. Thus, the search space sizes of the planners increase in the order SNLP* $\leq$ SNLP $\leq$ UA-SNLP $\leq$ TOCL.

To estimate the size of the search space explored under depth-first regimes, we need to estimate the probability of making successful choice. Suppose we assume that all the planners pick the next node to be refined randomly from the set of refinements that they generated. In terms of cost of depth-first search, the critical question is ‘‘what is the probability that each planner picks a refinement that contains at least one solution candidate.’’ We shall use the somewhat inaccurate term ‘‘success probability’’ to refer to this probability. Even small changes in this probability can have dramatic effects on performance since picking a wrong refinement at the very beginning of search can have very high impact on the performance of depth-first search.

Consider a search node $\mathcal{N}$ that SNLP* picked up for refinement. Suppose the average branching factor of SNLP's refinement strategy is b. SNLP will thus refine $\mathcal{N}$ into a set of b children nodes $\{\mathcal{N}_1, \mathcal{N}_2, \cdots, \mathcal{N}_b\}$. Since SNLP* is systematic, the candidate sets of these nodes define a partitioning of the candidate set of $\mathcal{N}$. Under depth first strategy, SNLP* will then pick one of the nodes, $\mathcal{N}_i$, for refinement.

Since SNLP, UA-SNLP and TOCL differ from SNLP* only in terms of additional refinements they employ (to provide tractable consistency check), we can look at their operation as further refining each $\mathcal{N}_i$ into some partition of its candidate set $\{\mathcal{N}_i^1, \mathcal{N}_i^2, \cdots, \mathcal{N}_i^m\}$ (as dictated by the pre-ordering/conflict resolution strategy), and selecting the next node to be refineed from this set.

It is clear that the success probability of SNLP, UA-SNLP and TOCL are connected to the success probability of SNLP* (since after all they are taking the refinements of SNLP and splitting them further). Suppose $\mathcal{N}_i$ contains m_i solution candidates. As long as pre-ordering partitions $\mathcal{N}_i$ into m_i or less refinements such that each refinement has a solution candidate, the success probability will tend to increase or remain the same. If on the other hand, $\mathcal{N}_i$ is split into more than m_i refinements, then the success probability will tend to decrease. (this means that any pre-ordering choice which splits nodes containing no solution candidates will tend to reduce the success probability). Even when $\mathcal{N}_i$ is split into less than m_i refinements, if the solution candidates are non-uniformly distributed such that not all refinements have solution candidates in them, then once again, the success probability will tend to decrease.

The above discussion shows that only when the pre-ordering/conflict resolution strategy is clairvoyant enough to split each $\mathcal{N}_i$ based on the number of solution candidates in the node, is it likely to increase the success probability. The pre-ordering/conflict resolution strategies used by SNLP, UA-SNLP and TOCL, use the constraints on the current partial plan, rather than knowledge about the number of solution candidates in a given node. Thus, they split the nodes in a somewhat uniform fashion that is oblivious to the number of solution candidates in the node. Thus, unless the domain is such that the syntactic splitting done by the pre-ordering strategies will have high correlation with the number of solution candidates in each node, there is no reason to expect that SNLP, UA-SNLP and TOCL will preserve or increase the success probability compared to SNLP*.

If we approximate the behavior of all the refinement strategies as a random partitioning of candidate set of $\mathcal{N}$ into some number of children nodes, then it is possible to provide a quantitative estimate of the success probabilities of all the three planners. To do this, let us assume as before that b is the average number of children nodes generated by the refinement strategy of SNLP*. Let us assume that to begin with $\mathcal{N}$ has m solution candidates in its candidate set. SNLP* partitions the candidate set of SNLP into b new refinements. The success probability of SNLP* 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 $q(m, n)$ is the binomial distribution:[25]

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

Now suppose that the average splitting factor of SNLP is s_s (i.e., SNLP splits each SNLP* node into s_u new nodes). This means that SNLP splits $\mathcal{N}$ into $b \times s_s$ nodes. The success probability of SNLP is thus $q(m, b \times s_s)$. Similarly, if we assume that s_u and s_t as the splitting factors for UA-SNLP and TOCL respectively, then their success probabilities are $q(m, b \times s_u)$, $q(m, b \times s_t)$ respectively. Since TOCL's preordering strategy is more eager than that of UA-SNLP, while UA-SNLP orders steps based on effects and preconditions, without regard to the causal links, we expect $s_s \geq s_u \geq s_t$.

It can be easily verified that for fixed m, $q(m, n)$ monotonically decreases with increasing n. Thus

$$q(m, b) \geq q(m, b \times s_s) \geq q(m, b \times s_u) \geq q(m, b \times s_t)$$

That is, under the random partitioning assumptions, the four planners, in the increasing order of success probability are: SNLP* $\geq$ SNLP $\geq$ UA-SNLP $\geq$ TOCL. (Although lower success

[25]$q(m, n)$ 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.

probability will suggest higher search space size, in terms of performance, the performance penalty of the larger search space size may still be offset by the reduced per-node refinement cost.)

6 Concluding Remarks

Much of the formalization work in classical planning concentrated on soundness and completeness of the planning algorithms. Very little comparative analysis has been done on the search space characteristics of the multitude of existing planning algorithms. We believe that the large part of the reason for this lack was the seemingly different vocabularies/frameworks within which many of the algorithms have been expressed.

In this paper, we have shown that if we take the view of planning as refinement search seriously, then we can indeed provide a generic refinement planning algorithm such that the full variety of classical planning algorithms can be expressed as specific instantiations of this algorithm. We have used our generic refinement planning algorithm to provide a coherent rational reconstruction of main ideas of classical planning algorithms. We have shown that this type of unified approach facilitates separating important ideas from ‘‘brand names’’ of the planning algorithms with which they have been associated in the past. The usual differentiations between planning algorithms such as ‘‘total order vs. partial order’’, ‘‘systematic vs. non-systematic’’, ‘‘causal link based vs. non-causal link based’’, ‘‘truth criterion based vs. secondary precondition based’’, and ‘‘TWEAK representation vs. ADL representation’’ have are all properly grounded within this unified refinement search framework. In many cases, the differences are shown to be a spectrum rather than dichotomy. All this, in turn facilitates a fruitful integration of competing approaches. We demonstrated this by providing a hybrid planning algorithm called UA-SNLP that borrows ideas from two seemingly different planning algorithms.

Our model of refinement planning clarifies the nature and utility of several oft-misunderstood ideas for bounding search space size, such as goal protection, protection intervals, and systematicity. We provided a clear formulation of the notion of systematicity and clarified several misconceptions about this property. Our model also provides a basis for analyzing a variety of ideas for improving planning performance, such as conflict deferment, deliberative goal selection strategies, least commitment, polynomial time consistency checks, etc., in terms of the tradeoffs between search space size and refinement cost that they offer. Because of our unified treatment, we were able to evaluate the utility of particular approaches not only for propositional STRIPS and TWEAK representations, but also under more expressive representations such as ADL. Our comparative analysis of the utility of pre-ordering and and conflict resolution techniques on the success probability of the planner show that the unified model also provides leverage in estimating explored search space in depth first regimes.

Our work also has important implications to the research on comparative analyses of partial order planning techniques. In the past, such comparative analyses tended to focus on a wholistic ‘‘black-box’’ view brand-name planning algorithms, such as TWEAK and SNLP (c.f. [10]). We believe that it is hard to draw meaningful conclusions from such comparisons since when seen as instantiations of our `Refine-Plan` algorithm, they differ in a variety of dimensions (see Table 1). A more meaningful approach, facilitated by the unified framework of this paper, involves comparing instantiations of `Refine-Plan` that only differ in 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 with SNLP, 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. After all, the fact that SNLP algorithm uses a protection-based termination check, and an arbitrary goal selection method, also does not mean that any instantiation of `Refine-Plan` that uses exhaustive causal links must use the same termination and goal-selection checks. In particular, it is perfectly feasible to construct a planning algorithm which uses MTC-based goal-selection and termination constraints like TWEAK, but employs exhaustive causal links (like SNLP). A comparison between *this* algorithm and TWEAK will clearly shed more light on the effect of exhaustive causal links and systematicity on planning performance. Similar experimental designs can be made for comparing the utility of pre-ordering strategies, conflict resolution strategies, termination criteria and goal selection criteria, based on the `Refine-Plan` algorithm template.

While refinement search provides a unified framework for many classical planning algorithms, it is by no means the sole model of planning in town. The so called transformational approaches to planning [13] provide an important alternative approach. The latter may be better equipped to model the HTN planners such as SIPE, NONLIN and OPLAN, which view planning as the process of putting together relatively stable canned plan fragments, and teasing out interactions between them, rather than as a process of starting with null plan and adding constraints.[26] On a related note, our current framework does not adequately account for the so-called state-based planners. We are currently working towards rectifying this situation.

The most significant contribution of our paper is that it provides a framework for foregrounding and analyzing a variety of tradeoffs within a single coherent refinement search framework. While our analysis has been qualitative in many places, we believe that this work provides the basis for further quantitative/empirical analysis of several tradeoffs. Exploration of these tradeoffs provides an important avenue for future research.

Acknowledgements

I benefited greatly from the (e-mail) discussions with David McAllester on the nature of general refinement search. Discussions with Bulusu Gopi Kumar helped significantly in clarifying my thinking on a variety of issues; most notably on the material in Section 5.5. Thanks are also due to

[26]Similar views are expressed by McDermott. In [13], he suggests that the plan-fragments used by hierarchical planners ought to be written in such a way that even with no further planning at all they can be conjoined with other plans (although the result may be suboptimal). The purpose of planning is then to remove the resulting inefficiencies.

Mark Drummond, Will Harvey, Laurie Ihrig, Suresh Katukam, Richard Korf, Pat Langley, Steve Minton, Ed Pednault, Mark Peot, Dave Smith, Austin Tate and Dan Weld for helpful comments during the course of this research.

References

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

[2] T. Dean, R.J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time and resources *Computational Intelligence,* Vol. 4, 381-398 (1988).

[3] S. Hanks and D. Weld. Systematic Adaptation for Case-Based Planning. In *Proc. 1st Intl. Conf. on AI Planning Systems*, 1992.

[4] W.D. Harvey and M.L. Ginsberg and D.E. Smith. Deferring Conflict Resolution Retains Systematicity. Submitted to AAAI-93.

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

[6] 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).

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

[8] S. Kambhampati and E. Cohen. On the utility of minimality-based pruning algorithms in partial order planning. ASU CSE Technical Report (In Preparation).

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

[10] 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: The Classical Approach and Beyond. March, 1993.

[11] P. Langley. Systematic and Nonsystematic search strategies. In *Proc. 1st Intl. Conference on AI Planning Systems*, June 1992.

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

[13] D. McDermott. Transformational planning of reactive behavior. Yale University TR CSD RR-941, December, 1992.

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

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

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

[17] D.S. Nau, V. Kumar and L. Kanal. General Branch and Bound, and its relation to A^* and AO^* *Artificial Intelligence*, Vol. 23, 1984, pp. 29-58.

[18] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishers, Palo Alto, CA, 1980.

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

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

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

[22] M.A. Peot and D.E. Smith. Threat-Removal Strategies for Nonlinear Planning. In *Proc. AAAI-93*

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

[24] A. Barrett and D. Weld. Partial Order Planning: Evaluating Possible Efficiency Gains Technical Report 92-05-01, Department of Computer Science and Engineering, University of Washington, Seattle, WA, June 1992.

[25] D.E. Smith and M.A. Peot. Postponing Simple Conflicts in Nonlinear Planning. In Working notes of AAAI Spring Symp. on Foundations of Automatic Planning: The classical approach and Beyond.

[26] G.J. Sussman. *A Computer Model of Skill Acquisition*. American Elsevier, New York, 1975

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

[28] Q. Yang, J. Tenenberg and S. Woods. Abstraction in nonlinear planning University of Waterloo Technical Report CS-91-65, 1991.

Contents

List of Figures