A Comparative analysis of Partial Order Planning and Task Reduction Planning

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

Abstract

Although task reduction (HTN) planning historically preceded partial order (PO) planning, and is believed to be more general than the latter, very little comparative analysis of the two planning formalisms has been done. Part of the reason for this has been the lack of systematic understanding of the functionalities provided by HTN planning over and above that of partial order planning. In this paper I will describe a generalized algorithm template for partial order planning based on refinement search, and extend it to cover HTN planning. I will use this framework as a basis to (i) discuss the similarities and differences between the HTN and the partial order planning methods, (ii) critically examine the claims regarding the efficiency and expressiveness of HTN planning, and (iii) shed light on several of the less understood features of HTN planning.

1 Introduction

Of late, there has been an increased interest in understanding the tradeoffs provided by the different classical planning algorithms, with the objective of forming predictive hypotheses regarding the fit between particular algorithms and problem types. Many recent research efforts have attempted comparative empirical analyses of planning algorithms [22, 1, 15, 13, 14]. One of the ironic things about all these analyses has been that the planning algorithms they consider, called partial order (PO) planning algorithms, differ from the algorithms used in many of the ''industrial strength'' classical planners, such as SIPE [17] and O-Plan [28, 27]. These planners use what is commonly called task-reduction planning or HTN (Hierarchical Task-Network) planning paradigm.

From a historical perspective, the lack of comparative work on HTN planning algorithms is quite puzzling. After all, the first big shift from state-space planning used in STRIPS [10] was not to partial-order planning, but rather to HTN planning, as used in NOAH [26]. Indeed, partial order planning, as it is understood today, is an off-shoot of Chapman's work on nonlinear planning [3] (which, ironically enough, was initially intended to be a formalization of task reduction planners such as NOAH, SIPE and NONLIN).[1] There are at least two general problems that lead to this state of affairs:

1. There has been very little formalization of HTN planning algorithms, with the result that for the uninitiated, it is very difficult to differentiate essential features from ''bells and whistles'' of HTN planning.
2. Although the affinity between HTN planning and partial order planning is well known, there is a lack of systematic understanding of the similarities and differences in functionality provided by the two planning paradigms. Some argue that HTN planners have substantial formal as well as practical advantages over partial order planners, while others have vtaken the position that HTN planning is an ''efficiency hack'' over partial order planning.

To deal with the first problem, recently Erol et. al. [6, 7, 8] have developed a coherent formalization and complexity analysis of HTN planning. At the same time, Barrett and Weld [2] and Young et. al. [32] have developed simpler implementations of HTN planners. Barrett and Weld's work has also taken some important first steps towards the development of sound and complete HTN planning for expressive action representations.

In this paper, I attempt to remedy the second problem, viz. the lack of comparative understanding of HTN and partial order planning algorithms. I will base my comparison on two generalized algorithm templates for partial order planning and HTN planning based on the idea of refinement search. Specifically, in my recent work [15, 13, 14], I have developed a generalized algorithm template for partial order refinement planning, the instantiations of which cover most existing PO planners. In this paper, I will extend that algorithm to cover HTN planners, and use the extended algorithm as a basis to do a careful analysis of the similarities and differences in the two planning paradigms.

We will see that the most important additional functionality offered by HTN planning (over partial order planning) allowing the user a greater control over the type of solutions generated by the planner. I will show that many of the perceived advantages of HTN planning over PO planning are related to this particular factor. I will also clarify and qualify many claims of expressiveness and efficiency that are attributed to HTN planners in the literature. Finally, the refinement search-based view of HTN planning developed in this paper will also help clarify several less understood/controversial features of HTN planning such as ''critics'', ''filter conditions'', and ''phantomization.''

Organization: The rest of this paper is organized as follows. Section 2, reviews the refinement search based semantics for partial order planning, and describes `Refine-plan-PO`, a generalized algorithm for partial order planning. Section 3 extends `Refine-plan-PO` to cover HTN planning and discusses the differences in the solution spaces and completeness characteristics of HTN and PO planners. Using these generalized planning algorithms as the background, Section 4 critically examines the various perceived advantages of HTN planning, including efficiency, expressiveness and applicability to realistic problems. Section 5 shows how refinement search based view of HTN planning also helps in clarifying several features of HTN planning such as condition typing, hierarchical promiscuity, and phantomization. Section 6 summarizes the contributions of this paper.

[*]This research is supported in part by NSF research initiation award (RIA) IRI-9210997, NSF young investigator award (NYI) IRI-9457634 and ARPA/Rome Laboratory planning initiative grant F30602-93-C-0039.

[1]As a historical note, Chapman [3] intended to ''clean'' up NOAH/NONLIN, and came up with the modal truth criterion which essentially extends the NONLIN Q&A criterion to deal with partial plans with variables. This version of plan-space planning, without task reduction, has come to be known as ''partial order planning''. Subsequently, Pednault [23] and McAllester [20], provided simpler formalizations of partial order planning without recourse to modal truth criteria, and sound and complete planners based on their formalizations have been implemented [1, 24, 12].

2 A unifying framework for Partial order planning based on Refinement Search

2.1 Plan Representation

A planning problem is a 3-tuple $\langle I, G, \mathcal{A}\rangle$, where I is the description of the initial state, G is the (partial) description of the goal state, and $\mathcal{A}$ is the set of actions (also called ''operators''). An action sequence (also referred to as ground operator sequence) S is said to `solve` a planning problem, if S can be executed from the initial state of the planning problem, and the resulting state of the world satisfies all the goals of the planning problem.

Given a planning problem, a planner attempts to generate an action sequence that `solves` the problem. Partial order planners do this by searching in the space of partial plans.[2] Partial plans are best seen as implicit representations for sets of ground operator sequences (potential solutions) [15, 13, 14]. In particular, a partial plan corresponds to a set of ground operator sequences that are consistent with the ordering, binding and auxiliary constraints on the plan. The operation of a partial order planner can be seen as that of repeatedly refining partial plans (i.e., adding constraints, and thus splitting their candidate sets) until a solution can be picked from the candidate set of the resulting partial plan.

A partial plan is a 5-tuple $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}\rangle$ where: T is the set of steps in the plan; T contains two distinguished step names t_0 and t_∞. $\mathcal{ST}$ is a symbol table, which maps step names to actions. The special step t_0 is always mapped to the dummy operator `start`, and similarly t_∞ is always mapped to `finish`. The effects of `start` and the preconditions of `finish` correspond, respectively, to the initial state and the desired goals of the planning problem. O is a partial ordering relation over T. $\mathcal{B}$ is a set of codesignation (binding) and non-codesignation (prohibited binding) constraints on the variables appearing in the preconditions and post-conditions of the operators. $\mathcal{L}$ is a set of auxiliary constraints that restrict the allowable orderings and bindings among the steps (see below).

An important type of auxiliary constraint is an **interval preservation constraint** (IPC), which is specified as a 3-tuple: $\langle t, p, t'\rangle$. A ground operator sequence S is said to satisfy an IPC $\langle t, p, t'\rangle$ of a plan $\mathcal{P}$,if there exists a mapping $\mathcal{M}$ between the steps of $\mathcal{P}$ and the elements of S, such that $\mathcal{M}$ is consistent with $\mathcal{ST}$ (i.e., each step t of the plan is mapped to the same action by $\mathcal{M}$ and $\mathcal{ST}$; $\mathcal{M}[t] = \mathcal{ST}[t]$) and every operator of S that comes between $\mathcal{M}[t]$ and $\mathcal{M}[t']$ preserves p. A second type of auxiliary constraint is the **point truth constraint** (PTC), which is specified as a 2-tuple: $\langle p, t\rangle$. A ground operator sequence S is said to satisfy a PTC $\langle p, t\rangle$ if there exists a mapping $\mathcal{M}$ between the steps of $\mathcal{P}$ and and the elements of S such that $\mathcal{M}$ is consistent with $\mathcal{ST}$ and p is true in the state preceding $\mathcal{M}[t]$ (The notion of truth here is that the condition is asserted by some operator and not deleted by any intervening operator).

Monotonicity of Auxiliary constraints: Auxiliary constraints can be divided into two classes, **monotonic** and **non-monotonic**. An auxiliary constraint is called **monotonic** if and only if once it is violated by a partial plan, no additional refinement of the partial plan will make it satisfy the constraint. Auxiliary constraints that do not have this property are called **non-monotonic**. According to this definition, the monotonicity of the auxiliary constraints depends on both the constraint and the nature of the refinements allowed. In particular, a constraint which is monotonic for a planner may be non-monotonic for another that employs a different set of refinements. IPCs are monotonic for normal partial order planning, while PTCs are non-monotonic. We will see that IPCs can be used to model bookkeeping (protection) constraints while PTCs can be used to model filter conditions and phantomization constraints (Section 5).

[2]For a more formal development of the refinement search semantics of partial plans, see [13, 15]

```
Algorithm Refine-Plan-PO($\mathcal{P}$) /*Returns refinements of $\mathcal{P}$ */
Parameters: sol: the routine for picking solution candidates from the candidate set of the partial plan 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.

0. Termination Check: If sol($\mathcal{P}$) returns a ground operator sequence that solves the problem, return it and terminate.

1.1 Goal Selection: Using the pick-open function, pick an open prerequisite $\langle C, t\rangle$ (where $C$ is a precondition of step $t$) from $\mathcal{P}$ to work on. Not a backtrack point.

1.2. Goal Establishment: Non-deterministically select a new or existing establisher step $t'$ for $\langle C, t\rangle$. Introduce enough constraints into the plan such that ($i$) $t'$ will have an effect $C$, and ($ii$) $C$ will persist until $t$. Backtrack point; all establishers need to be considered.

1.3. Bookkeeping: (Optional) Add auxiliary constraints noting the establishment decisions, to ensure that these decisions are not violated by latter refinements. This in turn reduces the redundancy in the search space. Bookkeeping strategies used by most existing planners can be modeled in terms of addition of interval preservation constraints.

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

    2.a. Pre-Ordering: Use some given static ordering mechanism, pre-order, to impose additional orderings between steps of the partial plans generated by the establishment refinement. Backtrack point; all interaction orderings need to be considered.
    2.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.

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

4. Recursive Invocation: Call Refine-Plan-PO on the the refined partial plan (if it is not pruned).
```

Figure 1: *A generalized algorithm for partial-order planning*

From a search control point of view, monotonic constraints are interesting because of the pruning power they provide. In particular, a partial plan can be directly pruned from search if none of its ground linearizations[3] satisfy a monotonic constraint. Non-monotonic constraints cannot be used for pruning (since a violated constraint may be satisfied after further refinement), but can be used as a basis for selection heuristics. For example, the planner may prefer partial plans that satisfy non-monotonic auxiliary constraints over those that do not yet satisfy them. This distinction will be used to explain several differences between HTN and Partial order planners in Section 5.

A plan is said to be **consistent with respect to an auxiliary constraint** c if at least one ground linearization of the plan satisfies the constraint. A partial plan is said to be **consistent** if at least one of its ground linearizations is consistent with respect to all the monotonic auxiliary constraints (such a ground linearization is called a **safe** ground linearization).

The candidate set of a partial plan is the set of all ground operator sequences that are consistent with the step, ordering and binding

[3]A **ground linearization** of a partial plan $\mathcal{P} : \langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}\rangle$ is a topological sort of T with respect to O, with all the variables instantiated to values that are consistent with the constraints in $\mathcal{B}$.

Planner	Soln. Constructor	Goal Selection	Bookkeeping	Tractability Refinements
Tweak [3]	MTC-based $O(n^4)$	MTC-based $O(n^4)$	None	None
UA [22]	MTC-based $O(n^4)$	MTC-based $O(n^4)$	None	Unambiguous ordering
Nonlin [27]	MTC (Q&A) based	Arbitrary $O(1)$	Interval & Goal Protection via Q&A	Conflict Resolution
TOCL [1]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Total ordering
Pedestal [18]	Protection based $O(1)$	Arbitrary $O(1)$	Interval Protection	Total ordering
SNLP [20] UCPOP [24]	Protection based $O(1)$	Arbitrary $O(1)$	Contributor protection	Conflict resolution
MP, MP-I [12]	Protection based	Arbitrary	(Multi) contributor protection	Conflict resolution
SNLP-UA	MTC based $O(n^4)$	MTC based $O(n^4)$	Contributor protection	Unambiguous Ordering
SNLP-MTC	MTC based $O(n^4)$	MTC based $O(n^4)$	Contributor protection	conflict resolution
McNONLIN-MTC	MTC based $O(n^4)$	MTC based $O(n^4)$	Interval protection	conflict resolution

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

constraints of the plan, and also satisfy its monotonic auxiliary constraints. A ground operator sequence is a solution candidate if it also satisfies the non-monotonic auxiliary constraints (if any), and `solves` the problem.

2.2 A generalized algorithm template for Partial Order Planning

As mentioned, the process of planning can be seen as a refinement search which starts with a null partial plan (whose candidate set corresponds to all possible ground operator sequences in the domain), and repeatedly refines it until a solution can be picked from the candidate set of a partial plan resulting from the refinements. Figure 1 provides a general algorithm template for partial order planning. I will now briefly describe the individual steps of the algorithm.

Termination and the Solution Constructor function: The job of a solution-constructor function is to look for and return a solution candidate that is consistent with the current constraints of the partial plan. Most existing solution constructors terminate when they reach a partial plan all of whose safe ground linerizations correspond to solutions for the planning problem.

Goal Selection and Establishment: The primary refinement operation in partial order planning is the so-called establishment operation. It selects a precondition $\langle C, s\rangle$ of the plan (where C is a precondition of a step s), and refines (i.e., adds constraints to) the partial plan such that different steps act as contributors of C to s in different refinements. Pednault [23] provides a general theory of establishment refinement for plans containing actions with conditional and quantified effects. Syntactically, each refinement corresponds to adding different sets of new step, ordering and binding constraints (as well as additional secondary preconditions, in the case of ADL actions [23]) to the parent plan.

Bookkeeping/Protection: Once a goal is established, many partial order planners employ bookkeeping strategies for remembering the specific establishment decision and protecting it during latter refinements. Bookkeeping strategies can be modeled in terms of adding auxiliary constraints to the partial plan. One popular bookkeeping strategy is the *interval protection*. Suppose the planner just established a condition c at step t with the help of the effects of the step t'. An interval protection strategy requires that no candidate of the partial plan can have p *deleted* between operators corresponding to t' and t. It can thus be modeled by adding an IPC (interval preservation constraint) $\langle t', p, t\rangle$. Another type of bookkeeping strategy, called *contributor protection*, requires that no candidate of the partial plan can have p either *added* or deleted between operators corresponding to t' and t. Thus contributor protection can be modeled by adding of the twin interval preservation constraints $\langle t', p, t\rangle$ and $\langle t', \neg p, t\rangle$. Finally, multi-contributor protections, such as those described in [12] can be modeled by adding a disjunction of IPCs.

Consistency check and Tractability Refinements: The consistency check is used to see if the partial plan is consistent with respect to all of its constraints (in particular, whether it has at least one ground linearization that satisfies all the auxiliary constraints). The consistency check turns out to be intractable for any partial plan containing IPCs (which, as we saw above, are added by most bookkeeping strategies). To make the consistency check tractable, many planners use secondary refinement operations called ‘‘tractability refinements’’ which push the complexity of consistency check into the search space. There are two types of tractability refinement strategies: *pre-ordering* and *conflict resolution*. In the case of pre-ordering, tractability of consistency check is achieved by restricting the type of partial orderings in the plan such that consistency with respect to auxiliary constraints can be checked without explicitly enumerating all the ground linearizations. In the case of conflict resolution, the partial plan is refined (by adding ordering and binding constraints) until each possible violation of the auxiliary constraint (called conflict) is individually resolved.

Table 1 characterizes many well-known partial order planning algorithms, as well as some novel ones as instantiations of `Refine-Plan-PO`. Because `Refine-Plan-PO` makes explicit the dimensions of variations among the existing planners, it facilitates normalized comparisons among them. In our recent work, we used this framework to conduct empirical studies to understand the fit between the domain characteristics and the performance of different types of partial order planners (corresponding to different instantiations of `Refine-Plan-PO`). Figure 2 contains a brief summary of these studies.

3 Task Reduction (HTN) Planning

Let us now see how `Refine-Plan-PO` can be extended to cover task reduction (HTN) planners. The partial plan representations used in HTN planning are similar to partial order planning with one important exception. Specifically, an HTN plan can be represented as a 5-tuple $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}\rangle$, with the exception that the steps in T are mapped to two types of actions (also called tasks): *primitive actions* which correspond to the usual actions in the PO planning, and *non-primitive (abstract) actions*. A necessary condition for the executability of a plan is that all steps be mapped to primitive tasks.

A generalized algorithm template for HTN planning is shown in Figure 3. Unlike the partial order planners, where the main refinement operation is the establishment operation, the main refinement operation in HTN planning is ‘‘task reduction.’’ This involves choosing a non-primitive task (say $t \in T$), and generating refinements corresponding to all possible ways of reducing t.

Comparative performance evaluation using Refine-Plan-PO

In the past, empirical analyses tended to focus on a wholistic ‘‘black-box’’ comparisons of brand-name planning algorithms, such as TWEAK vs. SNLP. It is hard to draw meaningful conclusions from such comparisons since, when seen as instantiations of our `Refine-Plan-PO` algorithm, they differ on a variety of dimensions (see Table 1). A more meaningful approach involves comparing instantiations of `Refine-Plan-PO` that differ only on a single dimension.

For example, from Table 1, we see that two of the important dimensions of variation among existing planners are the type and strength of protection strategies and tractability refinements employed by the individual planners. In [14, 13, 15], we developed qualitative models for the search space size and per-invocation cost of `Refine-Plan-PO`. According to this model, both tractability refinements and bookkeeping strategies can be seen as transferring complexity between search space size and per-invocation cost of `Refine-Plan-PO`. Tractability refinements attempt to reduce the cost of consistency check by refining partial plans further, and thereby increasing search space size. Similarly protection strategies attempt to reduce the redundancy in the search space by adding further auxiliary constraints to a partial plan, there by increasing the per-invocation cost of `Refine-Plan-PO`.

Since overall performance depends on both the per-invocation cost and the search space size, there is no reason to believe that any particular point the spectrum of possible tractability refinements and protection strategies will provide a uniformly dominant performance in all domains. The critical question instead becomes ‘‘*What are the characteristics of the domains where specific types of protection strategies/tractability refinements provide superior performance?*’’

To answer these questions, we conducted empirical studies with instantiations of `Refine-Plan-PO` that vary along the dimensions of tractability refinements and protection strategies [14, 15]. Our experiments show that the differences in tractability refinements cause greater performance differentials than the differences in bookkeeping strategies. Protection strategies affect performance only when the solution density is low, forcing the planner to explore a significant part of its overall search space. We also found that strong tractability refinements can lead to improved performance only when the additional constraints added to the plan by these refinements lead to a reduction of (simple) establishment possibilities. We demonstrate that this is most likely to happen in domains which have conditions that are added and deleted by many actions (we call such conditions ‘‘*high frequency conditions*’’).

Figure 2: Predicting the performance of instantiations of `Refine-Plan-PO`

```
Algorithm Refine-Plan-Htn($\mathcal{P}$) /*Returns refinements of $\mathcal{P}$ */
Parameters: pick-task: the routine for picking non-primitive tasks for
reduction plan to make conflict resolution tractable. auxiliary constraints.

0'. Termination Check: If all tasks of $\mathcal{P}$ are primitive, and if there
    is a safe ground linearization of $\mathcal{P}$ that solves the problem, return
    it and terminate.

1.1'. Task Selection: Using the pick-task function, pick an unre-
    duced task $t \in T$ from $\mathcal{P}$ to work on. Not a backtrack point.

1.2'. Task Reduction: Non-deterministically select a reduction
    schema $S : \mathcal{P}'$ for reducing $t$. Replace $t$ in $\mathcal{P}$ with $\mathcal{P}'$ (This
    involves removing $t$ from $\mathcal{P}$, merging the step, binding, ordering,
    symbol table and auxiliary constraints fields of $\mathcal{P}'$ with those of $\mathcal{P}$,
    and modifying the ordering and auxiliary constraints in $\mathcal{P}$ which
    refer to $t$ so that they refer to elements of $\mathcal{P}'$.
    Backtrack point; all reduction possibilities need to be considered

1.3'. Bookkeeping: Same as in Refine-Plan-PO

2'. Tractability Refinements: Same as in Refine-Plan-PO

4'. Recursive Invocation: Call Refine-Plan-HTN on the the re-
    fined partial plan (if it is not pruned).
```

Figure 3: *A generalized algorithm for HTN planning (only the steps that change from Refine-Plan-PO are shown)*

The reduction involves replacing a non-primitive task with a partial plan. The domain specification includes the legal ways of reducing individual non-primitive tasks. Suppose the task t in plan $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}\rangle$ is being reduced with the help of a reduction method

$$t \Rightarrow \mathcal{P}' : \langle T', O', \mathcal{B}', \mathcal{ST}', \mathcal{L}'\rangle.$$

(Notice that the plan fragment specified by the reduction method not only contains steps and orderings, but also auxiliary constraints, such as those corresponding to the IPCs and PTCs recommended in the reduction schema).

The resulting refinement is computed by removing t from $\mathcal{P}$, and substituting $\mathcal{P}'$ in its place, giving rise to a refined plan:

$$\mathcal{P}_R : \left\langle \begin{array}{c} \{T - t \cup T'\}, \{(O - O_t) \cup O' \cup O_m\}, \\ \{\mathcal{B} \cup \mathcal{B}' \cup \mathcal{B}'_m\}, \{\mathcal{ST} \cup \mathcal{ST}'\}, \\ \{(\mathcal{L} - \mathcal{L}_t) \cup \mathcal{L}'\} \end{array} \right\rangle$$

Notice that in replacing t with its reduction, we need to redirect any constraints that explicitly name t, to steps in its reduction. Thus, if O_t are the set of ordering constraints on $\mathcal{P}$ involving t, then they are removed, and a set of new constraints O_m are added in their place, such that the O_m contains orderings between steps of $\mathcal{P}$ and $\mathcal{P}'$. Similar redirection is also done for auxiliary constraints involving t. While HTN planning paradigm does not in general put any restrictions on how this redirection needs to be achieved, we will see in Section 5.3 that some properties of the planner (such as the ability to prune plans that are inconsistent with respect to the protection constraints, without losing completeness) depend on specific types of merging strategies (c.f. [31]).

A special type of non-primitive tasks are the so called *achievement tasks*. An achievement task $t : achieve(c)$ aims to make condition c true in the output situation of the task t. In addition to standard reductions, achievement tasks have **phantom** reductions, to handle the situations where the goal of the task is serendipitously achieved as a side effect of some other reduction. Given a task $t : achieve(c)$, a phantom reduction of t is a schema:

$$\mathcal{P} : \langle \{t'\}, \emptyset, \emptyset, \{t' \rightarrow \texttt{Noop}\}, \{\langle c, t'\rangle\}\rangle$$

Thus, t can be reduced by replacing it with a dummy task t' (which maps to a No-op), with the auxiliary point truth constraint that c is true in the situation preceding t' (see Section 2.1).

Observe that except for the main refinement step (which changes from precondition establishment to task reduction), the algorithms for PO and HTN planners are remarkably similar. It is interesting to note that many of the features of HTN planners, such as critics, and condition typing [17] can be accommodated without any major changes to `Refine-Plan-HTN`. In particular, critics can be accommodated as specialized tractability refinements and consistency

HTN Planning as a ''Problem''

Historically, HTN planning has been seen as a ''*method*'' rather than as a ''*problem*''. Our discussion in Section 3.1 brings up an alternative interpretation -- that HTN planners are solving a *different* planning problem compared to the partial order planners. I will now elaborate on this view.

Both partial order and state space planning methods aim to solve the so-called STRIPS planning problem. A STRIPS planning problem is a 3-tuple $\langle I, G, \mathcal{A}\rangle$ where I is the initial state, G is the final state, and $\mathcal{A}$ is the set of actions allowed in the domain. A ground operator (action) sequence S is said to be a solution if every action in S belongs to $\mathcal{A}$, and executing S in I produces a state of the world where G is satisfied. S is then said to `solve` the problem $\langle I, G, \mathcal{A}\rangle$.

The task reduction planners can be seen as solving a *different* problem, that we shall call the *HTN planning problem*. An HTN planning problem can be seen as a STRIPS planning problem augmented with a set of context-free grammars: $\langle I, G, \mathcal{A}, \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_m\}\rangle$. The terminal symbols in each of the grammars $\mathcal{G}_i$ are drawn from $\mathcal{A}$. An action sequence S is a solution to the HTN planning problem if and only if it `solves` the problem, and S is a valid sentence in each of the grammars in $\mathcal{G}$ [21, 7].

Since task reduction and partial order planners are solving different problems, the completeness criteria for these methods will naturally be different. In particular, a complete task reduction planner does not have to generate every ground action sequence that `solves` the problem, but only those which are also valid sentences in the grammars $\mathcal{G}_i$.

This problem-based characterization also explains the complexity difference between planning in STRIPS and HTN formalisms. Propositional STRIPS planning is known to be PSPACE-complete [20], whereas HTN planning is known to be undecidable in all but most restrictive cases [7]. As McAllester points out [21], this complexity difference can be understood on the basis of the fact that intersection of context free grammars (which is part of HTN planning problem) is an undecidable problem [11].

Figure 4: Formal characterization of the HTN planning problem

checks, while condition typing can be accommodated through the auxiliary constraints mechanism (see Section 5). Finally, some implementations of HTN planners, such as O-Plan [28] allow for both precondition establishment and task reduction refinements within the same algorithm. In this paper, I will concentrate on ''pure'' task reduction planners, and leave the issue of utility of combining both refinement strategies for future investigation.

Below, we will look at one important ramification of shifting from establishment refinement to task reduction refinement. In Section 5, we will discuss several additional ramifications of this shift.

3.1 Solution Space of HTN planners

As we mentioned earlier, in partial order planning, any ground operator sequence that `solves` the given problem is considered a solution to the problem. In contrast, not *all* such ground operator sequences are produced as solutions by a task reduction planner. Specifically, a ground operator sequence S that `solves` the problem is generated as a solution if and only if there is a way of reducing the initial null plan to S in terms of the reduction schemas provided to the planner. In [2], Barrett explains this in terms of the ''parseability'' of potential solutions in terms of the reduction schemas. From the refinement search point of view, task reduction refinements split the candidate set of a partial plan in a way that automatically prunes all ground operator sequences that will not have such a parse. Because of this, the solution space of HTN planners is different from that of partial order planners, and depends on the reduction schemas.

It is important to understand the practical utility of this difference in solution spaces. In many realistic planning problems, not every operator sequence that `solves` a problem may be an acceptable solution, as the users tend to have strong preferences about the *types* of solutions they are willing to accept. Consider the following two examples:

Example 1. A travel domain, where the user wants to eliminate travel plans that involve building an airport to take a flight out of the city (or even, stealing money to buy a ticket).

Example 2. A process planning domain with the task of making a hole, where there are two types of drilling operations, D1 and D2 and two types of hole-positioning operations H1 and H2. A hole can be made by selecting one hole-positioning and one hole-drilling operation. The user wants only those hole-making plans that pair D1 with H1 or D2 with H2.

In both the examples, the user is not satisfied with every plan that satisfies the goals, but only a restricted subset of them. Handling such restrictions in partial order planning would involve either attempting to change the domain specification (drop operators, or change their preconditions); or implementing complex post-processing filters to remove unwanted solutions. While the second solution is often impractical, the first one can be too restrictive. For example, one way of handling the travel example above is to restrict the domain such that the ''airport building'' action is not available to the planner. This is too restrictive since the there may be other ''legitimate'' uses of airport building operations that the user may want the planner to consider.

HTN planning provides the user a more general and flexible way of exercising control over solutions. In particular, by allowing non-primitive tasks, and controlling their reduction through user-specified task-reduction schemas, HTN planning allows the user to *control the planner's access to the actions in the domain*.

Another way of looking at the above is in terms of the solution languages defined by HTN and PO planners. The solution language of partial order planners is a regular language, while HTN planners also allow higher order solution languages (such as context free languages as well as their intersections) [7].[4] In particular, suppose the domain contains three actions $a1, a2, a3$. The solution plans produced by the partial order planners can be described by a regular language (such as $\{a1|a2|a3\}^*$), while that of HTN planners can be described by a higher level language (such as $a1^n a2^n a3^n$). Using partial order planners in domains with strong structural constraints is thus akin to using regular expressions to generate strings that are valid sentences in a context free languages -- although it can be done, it will be very inefficient. See Figure 4 for a discussion and elaboration on this view.

The difference in the solution spaces of HTN and PO planners has important ramifications. To begin with, the completeness of an HTN planners has to be defined with respect to both the domain actions, as well as the set of non-primitive tasks and the task reduction schemas (this is what Erol et. al. [8] do in their formalization of HTN

[4]The analogy between HTN task reduction schemas and operators in partial order planning on the one hand, and regular languages and Context Free Grammars on the other is first made by Erol [7].

planning). In the latter sections of this paper, we shall also see that this difference has implications to the efficiency of HTN planning.

4 Examining potential advantages of HTN Planning

Given the `Refine-Plan-Po` and `Refine-Plan-HTN` algorithm templates, we shall now address the claims regarding the advantages of HTN planners over PO planners. I will separate the claims regarding the advantages of HTN planning into three categories -- efficiency, expressiveness and applicability to realistic planning problems -- and critically examine each in turn.

4.1 Efficiency Arguments

The most important advantage claimed for HTN planning has to do with efficiency in plan generation. At first glance this seems counterintuitive -- after all, the worst case complexity of HTN planning is higher than that of STRIPS planning problem (see Figure 4)! Clearly, the efficiency claim needs to be qualified and circumscribed to those domains where the user does have strong biases about acceptable solutions. Once we consider this class of problems (which, as we argued earlier, are more representative of realistic planning problems), we can begin to understand the efficiency claims.

Pruning Power: To some extent, the very fact that HTN planning allows the users more control over specifying the types of solutions they want, also has ramifications on efficiency. After all, if the user did not want a travel plan that involves stealing money to buy a ticket, any effort the planner spends in refining such a plan is wasted. One question that still needs to be addressed is whether the ''task reduction'' approach is *required* to achieve this efficiency. Recently, Barrett and Weld [2] experimented with an interesting alternative approach. This approach uses HTN schemas to do incremental bottom-up parsing of the partial plans generated by a partial-order planner (UCPOP [24]), and prunes plans which do not have any parse. They compare this approach with task reduction approach and conclude that task reduction may still do better by avoiding even partial exploration of branches leading to solutions that violate the structural constraints required by the user.

(Re)using Canned Plans: Another source of efficiency for HTN planning is the fact that reduction schemas typically encode large plan fragments with pre-packaged causal structure. This obviates all the search needed to construct the plan fragment in the first place (as is done by a partial order planner), and shifts the focus from precondition establishment to merging of these canned plans. However, whether or not the use of canned plans improves efficiency depends on the level of interactions between the canned plans -- if they interact too much, the cost of merging could be as high as cost of synthesizing customized plans. Thus this source of efficiency depends critically upon the task reduction schemas of the domain being relatively interaction free.

A related issue is the differences between using task reduction schemas and doing HTN planning vs. using stored plans and facilitating plan reuse in the context of a partial order planning. To begin with, macro-operators typically do not have the hierarchical structure inherent in task reduction schemas. Further, macro operators and stored plans are always used along with primitive operators. In contrast, task reduction schemas are typically used *in lieu* of the primitive operators.[5]

Critics, Resources and Time Windows: Several other features, such as condition typing [27], time-windows [29] and resource based reasoning [28, 30] have been claimed to be sources of efficiency for task reduction planning. Although these ideas originated with HTN planners, they can also be used effectively in partial order planning. For example, time windows and resource reasoning aim to prune partial plans that are infeasible in terms of their temporal constraints and resource requirements. These can, in principle, be modeled as monotonic auxiliary constraints. In Section 5.1, we will discuss how filter conditions can be modeled with the help of auxiliary constraints.

Another feature of HTN planners that is said to make them more efficient is the use of ''critics.'' Within refinement search view of planning, critics can be modeled as generalized interaction detection and resolution procedures and/or consistency checks with respect to specific types of auxiliary constraints. Given that features such as resources, time windows and critics can be adapted to partial order planning as well as HTN planning, any argument about the relative efficiency of HTN planning that depends on these features has to be justified by the extra leverage, if any, achieved by the reduction levels in the planner.

4.2 Expressiveness arguments

Another class of advantages often associated with HTN planning are that of ''expressiveness.'' We have already discussed one type of expressiveness -- HTN planning provides the user the ability to specify arbitrary structural constraints on the acceptable solutions. The literature also contains claims of expressiveness in terms of the ability to model a larger class of planning problems and goals compared to PO planners; I will examine these in this section.

Intermediate Goals: *Intermediate goals* are useful in describing planning problems which cannot be defined in terms of the goal state alone. As an example, consider the goal of making a round trip from Phoenix to San Francisco. Since the initial and final location of the agent is Phoenix, this goal cannot be modeled as a goal of attainment, i.e., a precondition of t_∞ (unless time is modeled explicitly in the action representation [25]).

It has been mentioned in the literature (c.f. [6]) that such goals cannot be be modeled in classical planning without hierarchical task reduction. This claim needs to be qualified to some extent. Although much of the work on partial order planning concentrated on goals of attainment, it does not mean that other types of behavioral constraints cannot be handled within partial order planning. In most cases, allowing for a wider variety of goals simply involves starting with an initial partial plan that has more pre-specified constraints, and more dummy steps. For example, we can deal with the round trip goal within partial order planning by adding an additional dummy step (say t_D) to the plan such that t_D has a precondition $At(Phoenix)$ and t_∞ has a precondition $At(SFO)$, and $t_0 \prec t_D \prec t_\infty$.

What is harder is enforcing arbitrary structural restrictions (e.g. ''matching'' restrictions) on the different parts of the plan. An example would be enforcing a restriction that the two segments of the round trip should involve the same mode of transportation. While even this problem can be modeled and solved within a partial order planning framework, it cannot be done by merely starting with a more involved initial plan, and requires *changes to the domain*. HTN planners, on the other hand, can allow a more straightforward way of dealing with such problems by allowing the user to define a non-primitive task (say `Round-Trip(x,y)`) and associating customized reduction schema for that task.

Notice that the discussion in the preceding paragraph shows an alternative way of interpreting the expressiveness argument-- in terms of the control afforded to the user over the type of solutions generated by the planner.

[5]Although it is theoretically possible to make reduction schemas correspond to primitive operators, it is more likely that reduction schemas in realistic domains correspond to large plan fragments.

Looping and Iteration: Another expressiveness claim that is made in favor of HTN planning is regarding the ability to express looping and iteration. For example, suppose we want to model the task of emptying a truck. A natural way of doing this task is to keep removing one object at a time until the truck becomes empty. This can be modeled by having a non-primitive task called `Empty-truck`, and a reduction method:

`Empty-truck` $\Rightarrow$ [`take-out-widget` $\rightarrow$ `Empty-truck`].

Although task reduction schemas provide a natural way of modeling such looping [19], this claim needs to be qualified in two important aspects: (i) many types of problems involving looping can in fact be handled by partial order planners such as UCPOP [24] with the help of quantified goals, and (ii) very few implemented HTN planners actually are capable of dealing with non-trivial forms of looping. Elaborating on the first point, the truck example above can be easily modeled with the help of a quantified ''goal of attainment'' such as $\forall Object(x) \neg In(x, Truck)$. While UCPOP itself uses the static universe assumption and splits this goal into a large conjunctive goal, more recent partial order planners such as XII [9] also provide the capability to handle quantified effects in non-static universes (e.g., when the number of objects in the truck changes dynamically during plan execution). While there could be problems where the looping cannot be trivially converted into a quantified goal, it is not clear that any of the existing HTN planners are able to deal with such problems.

4.3 Applicability to ''realistic problems''

Perhaps the most controversial class of advantages claimed for HTN planning have to do with their ability to model ''realistic problems.'' Part of the reason for these claims has to do with the fact that the only large scale applications of domain independent classical planning have been done in the context of HTN planners. In [4], Drummond makes the argument that this is not a coincidence and that the planning task as well as the available knowledge in most realistic problems are such that they are best modeled in the HTN planning framework rather than the partial order planning framework. Specifically, he argues that the planning knowledge is in the form of relatively independent pre-formed abstract plans, rather than in the form of individual primitive actions. Further, he suggests that the planning activity in such domains is closer to plan merging than precondition establishment.

Our theoretical reconstruction of HTN planning is not particularly helpful in judging the validity of these claims. Instead their validity depends on the existence of a large number of ''stabilized'' domains where humans have already come up with the right types of relatively interaction free plan fragments. It is interesting to note McDermott's observations (made in the context of his comparison of regression planners and task reduction planners[6]) [18]:

> ''$\cdots$ The truth is that [regression] and [reduction] planners are not competing. The spaces searched by [reduction] planners are quite different from those searched by [regression] ones. A [reduction] planner pastes together big canned plans, postponing decisions about how those plans will interact. That approach makes no sense unless each of the plans is written in a robust way that will allow it to succeed when other things are happening. That gives the planner the freedom to ignore most interactions. In other words, the planner is not avoiding interactions by means other than search; instead, it is presupposing that plans have been written so that fatal interactions are improbable. This presupposition is false in the blocks world, where all the difficulties are due to intricate combinatorics in stringing together tiny pieces of plan.''

[6]Although McDermott used the terms ''linear'' and ''nonlinear,'' his ''linear'' planners correspond to regression planners which use precondition establishment as the main refinement operation and his ''nonlinear'' planners correspond to task reduction planners.

5 Clarifying features of HTN Planning

One of the important advantages of the representation and candidate set semantics of the partial plans developed in Section 2 is that it allows us to put in perspective many of the less-understood features of HTN planning, including filter conditions [5], abstraction vs. task reduction [17] and why plans with unresolvable protection conflicts cannot in general be pruned in HTN planning [31]. I will elaborate on this in the following three sections.

5.1 PTCs, Filter Conditions and Phantomization

An aspect of HTN planning that has been much misunderstood is the role of filter conditions (also called reduction assumptions) in planning. The implementors of HTN planners, including O-Plan [28] and SIPE [17] swear by them, while some other researchers have dismissed them as efficiency hacks that lead to incompleteness in partial order planners.

Filter conditions are the applicability conditions of the operators that should never be explicitly considered for establishment. Most previous work has characterized filter conditions as filtering out particular operators or task-reduction schemas from consideration. As has been pointed out by Pryor and Collins [5], using filter conditions this way will lead to loss of completeness.

In my view, the loss of completeness is a ramification of the erroneous interpretation (and implementation) of filter conditions. The right way to understand filter conditions is to see them as an integral part of the agenda to allow the user greater control over the types of solutions generated by the planner. In particular, filter conditions enable the domain writer to *disallow* certain types of solutions. For example, if the user wants to keep an airport building plan in the library, but disallow its use in a plan to fly from one city to another, this can be accomplished by making the existence of the airport a ''filter condition'' of the task of taking flight.

Given this understanding of filter conditions, it is fairly straightforward to include filter conditions into either of the refinement planning algorithms. Specifically, in refinement planning, filter conditions can be modeled as *point truth constraints* (see Section 2.1) of the form $\langle c, t\rangle$ with the semantics that the condition c must be true before t in every solution candidate. Any time an operator or a task reduction schema is selected, all the filter conditions are added as PTCs to the auxiliary constraints of the partial plan.

Since PTCs are in general non-monotonic auxiliary constraints, failure of filter conditions can not be used to prune partial plans. However, they can be used as a basis for selection heuristics. Specifically, we can always prefer partial plans which have filter conditions already established (this is not pruning). In fact, the use of filters in NONLIN, SIPE and other planners can be seen as a heuristic that considers all the plans with satisfied filter conditions first, before considering any plans with unsatisfied filters.

The preceding discussion about modeling filter conditions also has implications to the *phantomization* step in HTN planning (see Section 3). In planners like NONLIN [27], phantomization of a task $t : achieve(c)$ is accomplished by treating it as a *simple establishment*, and finding existing tasks t'' in the plan that can provide c. When such tasks are found, an IPC $\langle t'', c, t\rangle$ is added to the plan to remember the establishment relation. Since the simple establishment possibilities depend on the order in which the tasks are selected for reduction prior to t'', this method will provide different results for different task selection orders, thus necessitating backtracking on task selection order. The approach of adding a PTC, as done in `Refine-Plan-HTN`, (see Section 3), obviates this problem.

The discussion above also shows that filter conditions can not only be given clean semantics, they can be used whether we are using partial order or HTN planning paradigm. We also note see that the main functionality provided by filter conditions, viz., to provide user control over the type of solutions returned by the planner, is more in tune with the main functionality of HTN planners.

5.2 Abstraction levels vs. Reduction Levels

Given that task reduction is the main refinement operation of HTN planning, it is natural to try to attribute levels of ''abstractness'' to the various tasks in an HTN plan. Specifically, we can define the reduction level of a task in an HTN plan as the number of reductions that were done to introduce that task into the plan. Many of the ideas about HTN planning have been proposed with the implicit assumption that the reduction levels do correspond to abstraction levels. A necessary condition for any such correspondence is that no primitive task should ever be reduced to itself. It must be noted that once this restriction is enforced, the task reduction schemata cannot be used to model looping and iteration (see Section 4.2). Both the latter require schemata which allow a task to be recursively reduced to itself.

When there *is* a correspondence between reduction and abstraction levels, such a correspondence can be exploited in many ways to provide pruning power during planning. Consider an instance of `Refine-Plan-HTN`, where the task selection step always picks tasks of a higher level for reduction before it picks tasks of a lower level (note that this selection strategy is only one among the many possible selection strategies). For example, suppose a condition C occurs only as the effect of the set of tasks $T' \subseteq T$ (where T is the set of primitive and non-primitive task templates in the domain). Suppose further that there is a partial ordering among the tasks of T such that all tasks in T' are ordered to come at a higher level than that of some task t. Consider a situation where the planner is about to reduce t, and the reduction schema has a filter condition C. At this point, if C is not currently true in the plan, then we can safely prune the plan without worrying about loss of completeness. This can be seen in terms of monotonicity and non-monotonicity of auxiliary constraints: *the existence of a strict hierarchy among tasks (in terms of which tasks can be reduced by which) helps us turn a filter condition from a non-monotonic auxiliary constraint into a monotonic one.*

5.3 Admissibility of pruning plans with unresolvable conflicts

One of the differences introduced by changing step 1.2 in the `Refine-Plan` algorithm from establishment to task reduction refinement (see Figure 3) is that constraints that are normally monotonic for partial order planners can become non-monotonic. In particular, IPCs are a form of auxiliary constraints that are monotonic for partial order planners. (This is because once a plan is inconsistent with respect to a protection interval, no amount of step addition, ordering and binding constraints can make it consistent). However, IPCs can become non-monotonic in the presence of task reduction refinements. This is illustrated by the example in Figure 5. Specifically, a partial plan that is inconsistent with respect to its auxiliary constraints might become consistent after further task reduction. Thus, partial plans with unresolvable conflicts between their protection constraints cannot be pruned. This phenomenon was first noticed by Yang [31].

Given our reconstruction of HTN planning in terms of refinement search, this phenomenon can be explained easily. Since monotonicity of a constraint depends upon the types of refinements allowed by

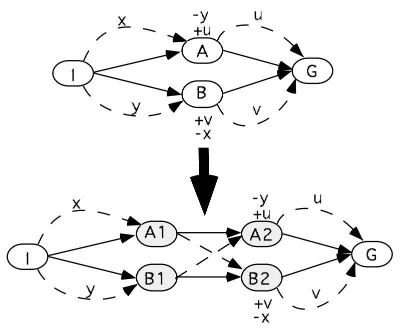

Figure 5: Example illustrating the non-monotonicity of IPCs when task reduction refinements are allowed. Although the top plan has no ground linearizations that are safe with respect to all the IPCs, the bottom plan, resulting from a reduction of A to A1 → A2 and B to B1 → B2, does have ground linearizations that are safe. The reason for this is that IPCs incident on and emerging from a single action are redirected to different actions after reduction (e.g. the IPCs incident on A are redirected to A1 while those emerging from from A are redirected to A2).

the planner, it is possible to lose monotonicity when the refinement is shifted from precondition establishment to task reduction. It is equally possible to regain monotonicity by placing restrictions on the refinement operations. Indeed, Yang [31] suggests a restriction called ''unique main subaction'' restriction, which effectively redirects all the auxiliary constraints involving a step t to a unique step t' in its reduction. With this restriction, IPCs become monotonic for task reduction refinements.

6 Summary and Conclusions

In this paper, I showed how a generalized algorithm for partial order planning can be extended to cover HTN planning. I have then used it to clarify various features of HTN planning, as well as to critically examine a variety of claims regarding the advantages of HTN planning. My analysis indicates that the primary advantage of HTN planners is the flexibility they provide the user in controlling the types of solutions generated by the planner. I argued that this flexibility has important ramifications on the efficiency of HTN planning.

I have also attempted to qualify and/or clarify several other claims about advantages of HTN planning. To summarize, the sources of efficiency particular to HTN planning stem from the use of reduction schemas -- these allow the planner to focus its search towards solutions that are acceptable to the user. The use of reduction schemas also allows the planner to shift the focus of the planner from localized precondition establishment to a more global plan merging. Other features such as critics, condition typing, while originating with HTN paradigm, are also applicable to partial order planners. In terms of refinement search, they can be seen as specialized consistency checks and tractability refinements. The main source of expressiveness for HTN planning is the ease with which user can place arbitrary structural constraints on acceptable solutions. Other claims about expressiveness -- including the ability to handle looping, intermediate goals etc. -- need to be qualified as they can also be handled in partial order planning to some extent.

Finally, I used the refinement search based model of HTN planning to re-interpret many features of HTN planning including the filter conditions, phantomization decisions, the relation between abstraction and reduction levels, and why plans with unresolvable protection conflicts cannot in general be pruned in HTN planning.

The work reported in this paper is by no means the last word on the comparative advantages of the two planning paradigms. In particular, comparative analysis will need focused empirical studies to understand which features of HTN planning will be useful in what types of domains. However, this paper does constitute a first step towards that latter goal. In particular, the understanding of the connections between HTN planning and partial order planning will mean that any insights regarding performance tradeoffs in partial order planning (e.g. [14, 22]) can be exploited in HTN planning. For example, the understanding of the effect of tractability refinements and protection strategies on performance, gained in the context of partial order planning (see Figure 2) can also be applicable in the context of HTN planning.

Acknowledgements

Thanks are due to Tony Barrett, Mark Drummond, Kutluhan Erol, Nort Fowler, James Hendler, David McAllester, Drew McDermott, Austin Tate, David Wilkins and the participants of the ASU planning seminar for constructive feedback and discussions.

References

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

[2] A. Barrett and D. Weld. Schema Parsing: Hierarchical Planning for Expressive Languages. In *Proc. AAAI-94*.

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

[4] M. Drummond. On precondition achievement and the computational economics of automatic planning. In *Proc. European Workshop on Planning Systems*, 1994.

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

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

[7] K. Erol, J. Hendler and D. Nau. HTN Planning: Complexity and Expressivity In *Proc. AAAI-94*.

[8] K. Erol, J. Hendler and D. Nau. UMCP: A sound and complete procedure for Hierarchical Task-network planning. In *Proc. AIPS-94*.

[9] K. Golden O. Etzioni and D. Weld. Omnipotence without Omniscience: Efficient sensor management for planning. In *Proc. AAAI-94*.

[10] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Readings in Planning*. Morgan Kaufmann, 1990.

[11] J.H. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation* Addison-Wesley Publishing, 1979.

[12] S. Kambhampati. Multi-Contributor Causal Structures for Planning: A Formalization and Evaluation. Arizona State University Technical Report, CS TR-92-019, July 1992. *Artificial Intelligence*, Vol. 69, 1994.

[13] S. Kambhampati. Refinement search as a unifying framework for analyzing planning algorithms. In *Proc. 4th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR-94)*, May 1994.

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

[15] S. Kambhampati, C. Knoblock and Q. Yang. Planning as Refinement Search: A Unified framework for evaluating design tradeoffs in partial order planning. ASU-CSE-TR 94-002. To appear in *Artificial Intelligence* special issue on Planning and Scheduling. 1995.

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

[17] D. Wilkins. *Practical Planning: Extending the classical AI Planning Paradigm* Morgan Kaufmann Publishers, San Mateo, CA (1988).

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

[19] D. McDermott. Planning and Acting. In *Readings in Planning*, Morgan Kaufmann, San Mateo (1990)

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

[21] D. McAllester. Private Communication. August 1994.

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

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

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

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

[26] E. Sacerdoti. *A structure for Plans and Behavior* Elsevier, North-Holland, New York (1977).

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

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

[29] S. Vere. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Trans. on Pattern Analysis and Machine Intell.*. Vol 5., pp 246-267 (1983).

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

[31] Q. Yang. Formalizing Planning Knowledge for hierarchical planning. *Computational Intelligence*, Vol 6, pp. 12-24 (1990).

[32] R.M. Young, M.E. Pollack and J.D. Moore. Decomposition and Causality in Partial-Order Planning. In *Proc. 2nd Intl. Conf. on AI Planning Systems*, 1994.