Appears in the working notes of AAAI-94 Workshop on Comparative Analysis of Planning Systems

# Comparing Partial Order Planning and Task Reduction Planning: A preliminary report

**Subbarao Kambhampati***
Department of Computer Science and Engineering
Arizona State University
Tempe AZ 85287-5406

## Abstract

Although task reduction (HTN) planning historically preceded partial order (PO) planning, and is understood to be more general than the latter, very little analysis has been done regarding its performance. 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. HTN planning has been characterized as everything from a panacea for the problems of partial order planners to a mere ''efficiency hack'' on partial order planning. In this paper I will extend a generalized algorithm for partial order planning, that I developed recent work, to cover HTN planning. I will use this as a basis to separate the essential and inessential differences between HTN and partial order planning.

## 1 Introduction

Of late, there has been a lot of 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 [12, 15, 14, 16, 1]. 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 [26, 25]. 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 [24]. 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 intended to be a formalization of planners such as NOAH, SIPE and NONLIN).[1] There are at least two reasons for this state of affairs:

- 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.
- While the affinity between HTN planning and partial order planning is well known, there is a lack of systematic understanding of the essential similarities and differences between them. While everybody agrees on what are the syntactic differences between them -- HTN planners use task reductions and partial order planners do not -- there is very little agreement on the 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 taken the position that HTN planning is an ''efficiency hack.''

Some preliminary work towards providing independent formal semantics for HTN planning has already been initiated by Erol et. al. [6, 7]. The current paper attempts to tackle these issues by reconstructing HTN planning starting from partial order planning.

Specifically, in my recent work [12, 15, 14], I have developed a generalized algorithm for partial order refinement planning. 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. In particular, I will distinguish three types of possible advantages of HTN planning over PO planning: Flexibility, Expressiveness and Efficiency. In each case, I will carefully examine the merits of the arguments in favor of HTN planning.

My analysis shows that the most important additional functionality offered by the HTN planners over PO planners is the increased flexibility that HTN paradigm provides for the

*This paper is also available as ASU CSE TR 94-001, March 1994. It reports on work in progress. Parts of it have developed out of the discussions in the ASU planning seminar. I benefited from the discussions with James Hendler, Austin Tate, Mark Drummond, Kutluhan Erol, and Drew McDermott. 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] As a historical note, Chapman intended to ''clean'' up NOAH/NONLIN, and wound up essentially recasting the Nonlin Q&A criterion, and losing in the process, the ideas of task reductions, protection intervals and critics. This version of plan-space planning, without task reduction, has come to be known as ''partial order planning''. Subsequently, Pednault [21] and McAllester [19], 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.

domain writer in terms of specifying the *types* of ground operator sequences that are accepted solutions for the problem. 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 the errors in some of the claims of expressiveness and efficiency that are attributed to HTN planners (by showing that those aspects can be easily duplicated in PO planners).

While parts of this paper are somewhat tutorial in nature, to the extent it succeeds in clarifying the similarities and differences between the two planning paradigms, I believe it will help both in capitalizing on the current comparative work on partial order planners, as well as in setting up more rigorous empirical comparisons among HTN planners in the future.

This paper is organized as follows. The next section, Section 2, very briefly reviews the preliminaries of classical planning, presents a generalized algorithm for PO planning, and extends it to cover HTN planning. This section shows that seen as refinement planners, the main difference between HTN planning and PO planning is the primary refinement they use: PO planners uses establishment refinement, while HTN planners use task reduction refinement. I will discuss how task reduction is a generalization of establishment refinement. Section 3 shows how the reconstruction of HTN planning provided in this paper also helps in clarifying several features of HTN planning such as condition typing, hierarchical promiscuity, and downward unlinearizability. Using the generalized planning algorithms as the background, Section 4 critically examines the various types of advantages of HTN planning. Section 5 summarizes the contributions of this paper.

# 2 HTN Planning vs. partial order planning -- the preliminaries

In this section, we will cast partial order planning and HTN planning within a uniform framework. We will start with partial order planning, discuss a plan representation and a generalized refinement planning algorithm. We will then discuss how the representation as well as the algorithm can be extended to cover HTN planning. We will start by reviewing the main objectives of the planning problem. Whatever the exact nature of the planner (partial order, state-space or HTN), the ultimate aim of (classical) planning is to find a *ground operator sequence*, which when executed in the given initial state, will produce desired *behaviors* or sequences of world states. In particular, a given ground operator sequence is said to `strips-solve` a planning problem, if the ground operator sequence can be executed from the initial state of the planning problem, and the sequence of states resulting from its execution satisfy all the goals of the planning problem. Most classical planning techniques have traditionally concentrated on the sub-class of behavioral constraints called the goals of attainment [11], which essentially constrain the agent's attention to behaviors that end in world states satisfying desired properties.

## 2.1 Partial order planning

Partial order planners search in the space of partial plans.[2] Partial plans are best seen as implicit representations for sets of ground operator sequences (potential solutions) [12, 15, 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 refinement planner can be seen as that of repeatedly splitting the candidate sets of the partial plan until a solution candidate can be picked up from the candidate set in a bounded time.

The following 5-tuple provides a uniform representation for partial plans that is applicable across all plan space planners: $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L} \rangle$ where: $T$ is the set of actions (step-names) in the plan; $T$ contains two distinguished step names $t_I$ and $t_G$. $\mathcal{ST}$ is a symbol table, which maps step names to domain operators. The special step $t_I$ is always mapped to the dummy operator `start`, and similarly $t_G$ is always mapped to `finish`. The effects of `start` and the preconditions of `finish` correspond, respectively, to the initial state and the desired goals (of attainment) of the planning problem. $O$ is a partial ordering relation over $T$. $\mathcal{B}$ is a set of codesignation (binding) and non-codesignation (prohibited bindings) constraints on the variables appearing in the preconditions and post-conditions of the operators. $\mathcal{L}$ is a set of auxiliary constraints that restrict the allowable orderings and bindings among the steps. An example of auxiliary constraint is the protection intervals, and contributor protection constraints used in many partial order planning algorithms.

Auxiliary constraints can be divided into two classes, *monotonic* constraints (or *candidate constraints*), and *non-monotonic* (or *solution constraints*). 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. A constraint is called non-monotonic if it is not monotonic.[3] Examples of monotonic constraints are the protection constraints employed by many planners (such as contributor protection and interval protection).

From a search point of view, monotonic constraints are interesting because of the pruning power they provide. In particular, any partial plan none of whose ground linearizations satisfy a monotonic constraint can be directly pruned from search. Solution constraints cannot be used for pruning, but can be used as a basis for selection heuristics. For example, the planner can give preference to partial plans that satisfy solution constraints already, over those that do not yet satisfy them.

A partial plan is said to be **consistent** if at least one of its ground linearizations is consistent with respect to all the auxiliary constraints. A ground linearization that is consistent with all auxiliary constraints is called a **safe** ground linearization. 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.

[2] For a more formal development of the refinement search semantics of partial plans, see [15, 12]

[3] The reason why we call the former candidate constraints and the latter solution constraints is that monotonic constraints must hold for every candidate of the plan, while solution constraints need only hold for the solution candidates.

**Algorithm Refine-Plan-PO($\mathcal{P}$)** /*Returns refinements of $\mathcal{P}$ */

**Parameters**: `sol`: the routine for checking termination `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 solution, return it and terminate.

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

**2.1. Goal Establishment:** Non-deterministically select a new or existing establisher step $s'$ for $\langle C, s\rangle$. Introduce enough constraints into the plan such that ($i$) $s'$ will have an effect $C$, and ($ii$) $C$ will persist until $s$. *Backtrack point; all establishers need to be considered.*

**2.2. Book Keeping:** (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. The auxiliary constraints may be one of goal protection, causal link protection or exhaustive causal link protection.

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

**3.a. Pre-Ordering:** 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.*

**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.*

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

**5. 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*

#### 2.1.1 Refinements in Partial Order Planning

The algorithm in Figure 1 provides a general outline for partial order planning algorithms. The primary refinement operation in partial order planning is the establishment operation -- pick a goal $\langle C, s\rangle$ where $s$ is a step of the plan and $C$ is a condition that needs to be true before $s$, and consider all possible ways of establishing it. Once a goal is established, many partial order planners add an auxiliary (book-keeping) constraints remembering the specific establishment decision. For most existing planners, these constraints can be expressed in terms of interval preservation constraints of the form $\langle s, p, s'\rangle$ with the semantics that the constraint is satisfied by a ground linearization of the plan as long as every step $s''$ that comes between $s$ and $s'$ preserves $p$. Finally, the consistency check checks 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).

Some planners that use book-keeping refinements also use what are called tractability refinements, to ensure that the partial plans continue to obey all the auxiliary constraints. In particular, the use of tractability refinements has the effect of pushing the complexity from the consistency check into the search space. Whether or not this improves the overall performance of the planner depends to a large extent on the way solutions in the candidate set of the partial plan get split by tractability refinements [15, 14].

### 2.2 HTN Planning

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 generic algorithm for HTN planning is shown in Figure 2. Unlike the partial order planners, where the main refinement operation is the establishment operation, the main refinement operation in HTN planning is what is known as a ''task reduction.'' This involves choosing a non-primitive task (say $t \in T$), and generating refinements corresponding to all possible ways of reducing $t$.

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 $\langle T, O, \mathcal{B}, , \mathcal{L}\rangle$ 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 ordering, but also auxiliary constraints, such as those corresponding to the protection intervals 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 3.1 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. [29]).

A special type of non-primitive tasks are the so called *achievement tasks* ($t : achieve(c)$), which aim to make condition $c$ true. The reduction of achievement tasks corresponds to the establishment refinement in PO planners. Achievement tasks can either be reduced with the help of appropriate task reduction schema (as done for other types of tasks), or can be handled by *phantomization*. In the latter case, the effects

**Algorithm Refine-Plan-Htn($\mathcal{P}$)** /*Returns refinements of $\mathcal{P}$ */
**Parameters**: `sol`: the routine for checking termination `pick-task`: the routine for picking non-primitive tasks for reduction `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 solution, return it and terminate.

**1. Task Selection:** Using the `pick-task` function, pick an unreduced task $t \in T$ from $\mathcal{P}$ to work on. *Not a backtrack point.*

**2.1. 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 $\mathcal{P}'$.

(If $t$ is a task of type $achieve(C)$, then one way of reducing $t$ is to establish $C$ with the help of some existing step in the plan; see Refine Plan). Introduce enough constraints into the plan such that ($i$) $s'$ will have an effect $C$, and ($ii$) $C$ will persist until $s$.

*Backtrack point; all reduction possibilities need to be considered.*

**2.2. Book Keeping:** (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. The auxiliary constraints may be one of goal protection, causal link protection or exhaustive causal link protection.

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

**3.a. Pre-Ordering:** 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*.

**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*.

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

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

Figure 2: *A generalized algorithm for HTN planning*

of a step $t'$ in the plan are used to establish $c$ at $t$ (and an appropriate book-keeping constraint is added to the plan), and the type of $t$ is replaced with a dummy ''No op'' primitive task.

Note that except for the establishment step (and the change from goal selection to task selection in Step 1), 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, condition typing can be accommodated without any major changes to `Refine-Plan-HTN`. In particular, critics can be accommodated as tractability refinements (step 3), while condition typing can be accommodated through the auxiliary constraints mechanism (see Section 3).

# 3 Clarifying some features of implemented HTN Planners

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 features of HTN planning, including filter conditions [5], abstraction vs. task reduction [17] and downward-unlinearizability [29].

In particular, a proper understanding of auxiliary constraints clarifies several misconceptions about the HTN planning. We will start by recalling the following points about auxiliary constraints: The auxiliary constraints give pruning only when they are monotonic (but can be used as selection heuristics if they are non-monotonic). Whether or not an auxiliary constraint is monotonic depends on the type of refinements done by the planner (see the definition in Section 2.1). Finally, we can change the nature of a constraint from a monotonic to non-monotonic one by restricting the type of refinements, or constraining the type of domain theory. We can use this understanding to clarify several issues in HTN planning.

## 3.1 Downward Unlinearizability

One of the differences introduced by changing Step 2.1 from establishment to task reduction refinement is that constraints that are normally monotonic for partial order planners can become non-monotonic. In particular, protection intervals 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, protection intervals can become non-monotonic in the presence of task reduction refinements. Specifically, a partial plan that is inconsistent with respect to its auxiliary constraints might become consistent after further task reduction. Thus, inconsistency with respect to protection intervals cannot be used to prune partial plans. Yang [29] noticed this phenomenon first, and developed certain restrictions on the way auxiliary constraints are redirected during task reduction (Section 2.2)[4], under which the plans inconsistent with respect to protection intervals can be pruned without loss of completeness.

In terms of our reconstruction of HTN planning, this phenomenon can be explained easily. Since monotonicity of a constraint depends upon the types of refinements allowed by the planner, it is possible to lose monotonicity when additional refinement operations are added to the planner. It is equally possible to regain monotonicity by placing restrictions on the refinement operations.

## 3.2 Filter Conditions/Reduction Assumptions

One of the aspects of HTN planning, that have been much misunderstood is the role of Filter conditions/Reduction assumptions in planning. The writers of HTN planners, including Oplan [26] and SIPE [17] swear by them, while some

[4] Yang [29] phrases his restrictions in terms of the types of the reduction schemata allowed in the domain. It can however be recast in terms of restriction on the way protection intervals are redirected during task reduction

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. Filter conditions thus provide a way for the domain writer to *disallow* certain types of solutions (e.g., building an airport in a city for the express purpose of going from there to another city) even if they satisfy the standard definition of plan solutions.[5]

Given this understanding of filter conditions, it is fairly straight forward to include filter conditions into either of the refinement planning algorithms. Specifically, in refinement planning, filter conditions should be seen as point truth constraints that need to hold in every solution candidate. In particular, they can be modeled as point truth constraints of the form $\langle c@s \rangle$ with the semantics that the condition $c$ must be true before $s$. Any time an operator or a task reduction schema is selected, all the filter conditions are added as point protections to the auxiliary constraints of the partial plan.

Most previous work has characterized filter conditions as filtering out particular operators or task-reduction schemas from consideration. Unless the auxiliary constraints representing filter conditions are monotonic, this type of pruning can lead to loss of completeness. Some researchers (c.f. [5]) have suggested that unless filter conditions are used to prune, they do not affect the efficiency of planner in any way.

In my view, the primary functionality of filters is not to improve efficiency, but to enable the user to disallow certain types of solutions. Secondly, while filter conditions can not be used to prune partial plans, 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.[6]

The discussion above also shows that filter conditions can not only be given clean semantics, they can be used whether or not we are using partial order or HTN planning paradigm. Later in the paper (Section 4.1), we will 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.

### 3.3 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, reduction level of a task in an HTN plan is the number of reductions that were done to introduce that task into the plan. From the beginning, one of the questions about HTN planning has been whether or not the reduction levels correspond to abstraction levels. 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 require schemata which allow a task to be recursively reduced to itself.[7]

When there *is* is a correspondence between reduction and abstraction levels, such a correspondence can be exploited in many ways to provide pruning power during planning. Suppose that the task selection step always picks tasks of a higher level for expansion 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 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 $t$. Now suppose 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. Once again, this can be seen in terms of monotonicity and non-monotonicity of auxiliary constraints: the existence of strict hierarchy among tasks (in terms of which tasks can be reduced which) has helped us turn a non-monotonic constraint into a monotonic one.

## 4 Examination of the advantages of HTN Planning

Given the many similarities between the partial ordering and HTN planning paradigms, the next question I would like to address is regarding the specific advantages (such as additional functionalities provided) of HTN planners over PO planners. In the past, many different claims have been made in this regard. I will separate the claims regarding the advantages of HTN planning into three categories and critically examine them in turn:

**Flexibility:** Claims regarding how HTN planning methodology provides more flexibility to the user in modeling the planning domain and problem.

**Expressiveness:** Claims that HTN planners can handle a larger class of goals and problems than PO planning.

**Efficiency:** Claims that HTN planers are more ''efficient'' in plan generation than PO planners.

[5] Notice, once again, the similarity between filter conditions and the control over solutions provided by HTN planning; see Section 4.1.

[6] In fact, even Pryor and Collins[5] do use filter condition satisfaction as part of the ranking metric in one of their experiments, and find that it substantially improves the performance of a partial order planner. Pryor and Collins also argue that the functionality of filter conditions can be achieved through use of conditional effects and secondary preconditions. As the foregoing discussion shows, this statement is incorrect. Secondary preconditions can not be used to replace filter conditions. The semantics of secondary preconditions allow explicit subgoaling on these preconditions (for example, to make a step with a conditional effect ''*If P then C*'' establish a condition $C$, we may subgoal on the secondary precondition $P$. In contrast, one is never allowed to achieve filter conditions. Thus if the same operator had a filter condition $P$ and an effect $C$, we cannot subgoal on $P$ to establish $C$. Any solution in which $P$ does not hold will be disallowed. While it may be true that some planners mistakenly used filters to get the functionality of context dependent effects, in general, secondary preconditions and filters have very different semantics.

[7] Erol et. al. [7] show that when there is a strict level associated with the tasks in the domain, then HTN planning is decidable.

### 4.1 Flexibility Arguments

HTN planning has been considered to be more ''user-flexible'' and user-friendly than partial order planning. The arguments about flexibility are to some extent subjective and have never been stated formally. In this section, I will separate and discuss one type of flexibility, that I think is a major advantage of HTN planning. It is the flexibility they provide the users and domain-writers to delineate the types of solutions that they want the planner to produce.

To elaborate, note that the definition of solutions in terms of `strips-solve` in (see Section 2) means that any ground operator sequence that is executable and satisfies the goals of the problem is a valid solution. Unfortunately however, not all such solutions are reasonable. In realistic domains, the users may 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 alternately, 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 implement 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 access of the planner to the actions in the domain.[8]

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). In particular, suppose the domain contains two actions $a1$ and $a2$. The solution plans produced by the partial order planners can be described by a regular language (such as $\{a1|a2\}^*$), while that of HTN planners can be described by a context free language (such as $a1^n a2^n$).[9]

The fact that HTN planners allow the user to put restrictions on the types of solutions returned by the planner also has some fundamental ramifications on the notions of completeness of the planner. We know that the completeness of a partial order planner can be determined given the actions. In particular, given a planning problem in the domain, if there is any ground operator (action) sequence that can `strips-solve` the problem, then a complete partial order planner must be able to find a solution for it.

In the case of HTN planners, the fact that there exists a ground operator sequence that `strips-solves` the problem does not necessarily mean that a complete planner must find it as a solution. In particular, unless there is a sequence of reductions of the goals tasks that results in that ground operator sequence, the HTN planner will not find it. Thus, the completeness of an HTN planner 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).

### 4.2 Expressiveness arguments

An important class of advantages often associated with HTN planning are that of ''expressiveness'', or the ability to model a larger class of planning problems compared to PO planners. While for the most part these comments have been made informally (e.g., in conference panels), some of them have also appeared in the literature (e.g. [6]). In this section, I will examine the expressiveness arguments.

**Intermediate Goals:** *Intermediate goals* are useful to describe planning problems which cannot be defined in terms of the goal state alone. As an example, consider the goal of making a round trip from Phoenix to San Francisco. 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 $t_\infty$ (unless time is modeled explicitly in the action representation [23]).

It has been mentioned in the literature (c.f. [6]) that such goals cannot be be modeled in classical planning without hierarchical task reduction. To some extent, this claim is erroneous and needs to be qualified. 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 withing partial order planning. In most cases, allowing for a wider variety of goals simply involves starting with an initial partial plan $\mathcal{P}_\emptyset$ 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 $t_\infty$ has a precondition $At(SFO)$, and $t_0 \prec t_D \prec t_\infty$.

What is harder is enforcing ''matching'' restrictions on the different parts of the plan. For example, enforcing a restriction that the two segments of the round trip should

[8] In [4], Drummond makes a similar point, suggesting that task-reduction schemas allow the user to not only specify what skeleton plan is used to achieve a goal, but also to specify *a priori* what causal dependencies must hold in the plan fragment. He observes that this shifts the focus from precondition establishment to plan merging.

[9] To my knowledge, 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]. However, Erol seems to use the analogy to explain the complexity of HTN planning, rather than to explain the types of functionalities it provides).

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-- it is not that HTN planners are able to deal with more expressive goals, but that they allow the users more control over the kinds of solutions they are willing to accept (see Section 4.1).

**Looping and Iteration:** Another expressiveness claim that is made in favor of HTN planning is regarding the ability to express looping and iteration. From the very beginning [18], it has been noted that task reduction schemata provide a natural way to model looping. 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 done by having a non-primitive task called `Empty-truck`, and a reduction method:

`Empty-truck`

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

The problem with these arguments is two fold: ($i$) many types of problems involving looping can in fact be handled by partial order planners such as UCPOP [22] 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 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 planners are able to deal with such problems.[10]

### 4.3 Efficiency Arguments

Another class of advantages claimed for HTN planning deal with efficiency in plan generation. 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.

Another source of efficiency for HTN planning is the fact that reduction schemas typically encode large plan fragments that are relatively stable. This obviates all the search needed to construct the plan fragment in the first place (as is done by a first-principles based partial order planner).[11]

It may be argued that this particular efficiency can be achieved within the partial order planing framework through techniques such as macro-operators. Although the use of canned plans in HTN planning is reminiscent of the macro-operator type techniques for improving planning perfomance, there are some important differences. To begin with, macro operators typically do not have the hierarchical structure inherent in task reduction schemas. Further, macro operators are always used along with primitive operators. Thus the macro-operators tend to increase the branching factor of the search, reduce solution depth, but leave the completeness (with respect to primitive operators) unaffected.

On the other hand, task reduction schemas are typically used in lieu of the primitive operators.[12] Thus, the branching factor does not necessarily increase in the presence of task reduction schemas. Thus, while the depth can reduce and branching factor does not necessarily increase. Of course, the completeness with respect to primitive operators is lost. However, as we discussed earlier, this should be seen as a *feature* rather than a *bug*, in that reduction schemas provide the user the flexibility to prune specific classes of plans.[13]

Several other features, such as condition typing [25], time-windows [27] and resource based reasoning [26, 28] have been claimed to be sources of efficiency for task reduction planning. efficiency in HTN 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 in terms of monotonic auxiliary constraints. We have already discussed 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.'' Critics are generalized interaction detection and resolution procedures. The operation of most of the ''critics'' used in the implemented planners can be formalized as doing tractability refinements 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

[10] In fact, many existing HTN planners attempt to map reduction levels and abstraction levels. We already noted (see Section 3.3) that recursive schemas that are needed to model looping will make it harder to enforce such a mapping.

[11] Of course, there is still the search involved in dealing with interactions that arise in merging the different plan fragments.

[12] 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.

[13] Barrett and Weld [2] provide an interesting alternative approach for exploiting the efficiency aspect of task reduction schemas. They use HTN schemas to do incremental bottom-up parsing of the partial plans generated by a partial-order planner (UCPOP [22]), and prune any plans which do not have any parse. An interesting open question is to what extent the functionality of HTN planning can be achieved through a partial order planner augmented with a schema-parser. Clearly, the solution control aspect of HTN planning can be handled through schema parsing to some extent. However, it is not clear how schema parsing will be able exploit the features of HTN planning that depend on the levels of detail in the plan.

the planner.

## 5 Summary

In this paper, I showed how a generalized algorithm for PO planning can be extended to cover HTN planning. I have then used it to clarify the various features of HTN planning, as well as 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 observed that this flexibility can also have ramifications on the efficiency. I have also attempted to qualify and/or clarify several other claims about advantages of 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, I believe this does constitute a first step towards that 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, 20]) can be exploited in HTN planning. It could also help us in formulating focused empirical studies to understand the tradeoffs provided by the various features of HTN planners.

## References


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

[2] A. Barrett and D. Weld. Schema Parsing: Hierarchical Planning for Expressive Languages. Submitted to 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* (to appear)

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

[9] K. Golden O. Etzioni and D. Weld. Omnipotence without Omniscience: Efficient sensor management for planning. Submitted to AIPS-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] M.G. Georgeff. Planning. In *Readings in Planning*. Morgan Kaufmann, 1990.

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

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

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

[15] 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.

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

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

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

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

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

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

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

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

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

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

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

[27] 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).

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

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