Toward a General Framework for Hierarchical Task-Network Planning (Extended Abstract)*

Kutluhan Erol† Dana Nau‡ James Hendler§
Department of Computer Science
and Institute for Systems Research
University of Maryland
College Park, MD 20742

Introduction

In AI planning research, planning practice (as embodied in implemented planning systems) tends to run far ahead of the theories that explain the behavior of those planning systems. For example, the past few years have seen much analysis of the properties of totally- and partially-ordered planning systems using STRIPS-style planning operators (Minton *et al.*, 1991; McAllester and Rosenblitt, 1991; Chapman, 1987). However, most of the practical work on AI planning systems for the last fifteen years has been based on hierarchical task-network (HTN) decomposition(Sacerdoti, 1990; Tate, 1977; Wilkins, 1984) as opposed to STRIPS systems, yet there has been very little similar analytical work on the properties of hierarachical task-network (HTN) planning systems.

One of the primary obstacles impeding such work has been the lack of a clear theoretical framework explaining what a HTN planning system is. A primary goal of our current work is to correctly define, analyze, and explicate features of the design of HTN planning systems. In this report, we describe some first steps toward that goal: We set out a formal definition of HTN planning, present a nondeterministic HTN planning procedure $\mathbf{P}_{HTN}$ that is general enough to include most existing HTN planning procedures as special cases, and present theorems showing that $\mathbf{P}_{HTN}$ is sound and complete.

In addition, we present theorems showing that if $\mathbf{P}_{HTN}$ is run on planning problems P_1 and P_2 that equivalent except that P_2 has "more informed" methods and/or critics, then $\mathbf{P}_{HTN}$ will perform more efficiently on P_2. These results are analogous to the informedness results for the well-known A* search procedure.

*This work was supported in part by NSF Grants NSFD CDR-88003012 and IRI-8907890.

†kutluhan@cs.umd.edu
‡nau@cs.umd.edu
§hendler@cs.umd.edu

A General Definition of HTN Planning

In this section, we outline a formal definition of hierarchical task-network (HTN) planning. We omit a number of details here; for more detailed definitions see (Erol *et al.*, 1993).

HTN planning systems work with symbolic representations of the problems they are trying to solve. By a *planning language*, we mean the language $\mathcal{L}$ in which these problems are represented. Basically, $\mathcal{L}$ is a first-order language generated by finitely many constant symbols, predicate symbols, and function symbols—but to represent the tasks that the planner needs to achieve, we augment $\mathcal{L}$ to include a a special symbol **achieve** and a finite set of *task symbols*, which are used to form *tasks* as we will describe below.

A *state* is a finite set of ground atoms[1] in $\mathcal{L}$. Intuitively, a state tells us which ground atoms are currently true: if α is a ground atom, then α is true in state s if and only if $\alpha \in s$. For example, suppose I am in Union Station in Washington, DC, and have enough money to buy a train ticket to Penn Station in New York City. I might represent this with the following state:

$$s_1 = \{\text{in(U)}, \text{have-money()}\},$$

where U is a constant symbol representing Union Station. A *task* is either of the following:

1. A task symbol followed by a list of terms. For example, to represent the objective of taking a round trip by train between Union Station and Penn Station, one might use the following task:

   $$t_1 = \textbf{round-trip}(\text{U}, \text{P}),$$

   where U is as before, and P is a constant symbol representing Penn Station.

2. The expression **achieve**[α], where α is any atom. For example, to represent the objective of having a train ticket from x to y, one might use the task

   $$t_2 = \textbf{achieve}[\text{have-ticket}(x, y)].$$

[1] An atom is a predicate symbol followed by a list of terms.

As another example, the objective of having a ticket from Union Station to Penn Station would be represented by the task

$$t_2\theta_1 = \textbf{achieve}[\text{have-ticket}(x, y)],$$

where θ_1 is the substitution

$$\theta_1 = \{\text{U}/x, \text{P}/y\}.$$

A *planning operator* o is either of the following:

1. A pair (Name(o), Effects(o)), where
   - Name(o), the *name* of o, is a syntactic expression of the form $o(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are variable symbols;
   - Effects(o), the *effects* of o, is a set $\{c_1, \ldots, c_k\}$, where each c_i is a set of literals[2] that contains no variables other than $x_1, \ldots, x_n$.
2. A substitution instance $o\theta$ of another planning operator o. If $o\theta$ contains no variables, then we call it a *ground operator.*

Note that we do not include preconditions in our planning operators; these will be handled in the *methods* defined below.

Suppose s is a state and o is a ground operator. Then the *result* of applying o to s is the state

$$o(s) = (s - \{\text{all negative literals in Effects}(o)\}) \cup \{\text{all positive literals in Effects}(o)\}.$$

Intuitively, $o(s)$ is the state resulting from s if we perform the action represented by o.

For example, to represent the action of purchasing a train ticket from location l_1 to location l_2, we might use the following planning operator:

Name: BUY-TICKET(l_1, l_2)
Effects: {¬have-money(), have-ticket(l_1, l_2)}

Clearly, the atom have-money() is too simple-minded to represent fully the cash-flow effects of paying for the ticket. To correctly handle resource usage requires some extensions to the framework. We intend to deal with these extensions in our future work.

A *plan* is a totally ordered set of ground operators.

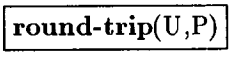
Intuitively, a *task network* is a graph structure representing a partially or fully developed plan. More formally, we will define it to be a pair $N = (G, C)$, where[3]

1. $G = (T, \prec)$ is a partially ordered set of tasks and operators, where T is the set and $\prec$ is the partial ordering. Usually we will represent G by drawing a directed graph, and for this reason we call G the *graph* of N.
2. C is a set of constraints called the *constraint set* of N. These constraints might involve variable bindings, or they might be *protected intervals*, i.e., expressions of the form (n_1, n_2, e), where $n_1, n_2 \in T$ and e is a statement in $\mathcal{L}$, meaning that in any execution of any plan that is a solution for N, e should be true in all states between n_1 and n_2. A special case of this constraint is when $n_1 = n_2$. It means e should be true at the state immediately before n_1. (To understand where these states come from, see the definition of the solution set $sol(N)$ below.)

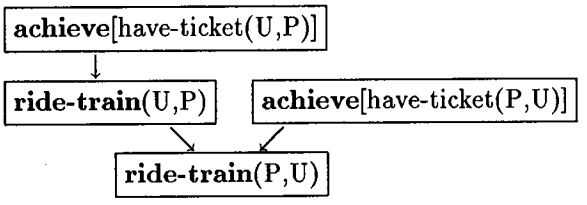
Note that if P is a plan, then $(P, \emptyset)$ is a task network.

As examples, each of the following is a task network whose constraint set is empty:

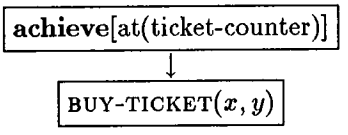
$N_1 =$



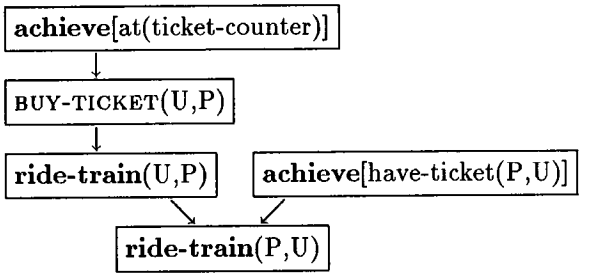
$N_2 =$



$N_3 =$



$N_4 =$



A *method* is an ordered pair $m = (t, N)$, where t is a task and N is a task network. Informally, this means that N is a possible way to achieve t. For example, to indicate that N_2 and N_3 are ways to achieve the tasks t_1 and t_2 discussed earlier, we can define the following methods:

$$m_1 = (t_1, N_2);$$
$$m_2 = (t_2, N_3).$$

Suppose N is a task network, t is a task in N, $m = (t', N')$ is a method, and θ is a substitution that unifies

[2] A literal is either an atom (in which case we say the literal is *positive*), or the negation of an atom (in which case we say the literal is *negative.*

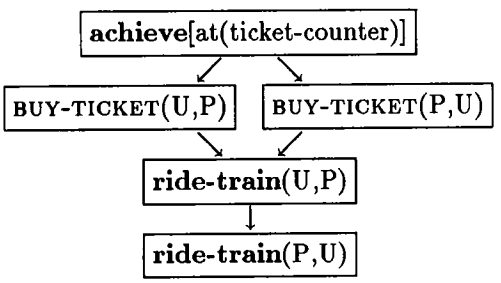
[3] Some implementations of hierarchichical task networks also associate effects with N. We have left them out in the current formalism, because we believe we can handle them using constraints. We will investigate this issue more fully in our future work.

t and t'. Then we say that m *matches* t, and we define reduce(N, t, m, θ) to be the task network obtained from $N\theta$ by replacing $t\theta$ with the graph of $N\theta$, and incorporating $N\theta$'s constraint set into N's constraint set. This task network is called a *reduction* of N. Here are some examples of reductions:

$$\begin{aligned} \text{reduce}(N_1, t_1, m_1, \emptyset) &= N_2; \\ \text{reduce}(N_2, t_2\theta_1, m_2, \theta_1)] &= N_4; \end{aligned}$$

where $\theta_1 = \{\text{U}/x, \text{P}/y\}$ as defined earlier.

A *critic function* is a function κ that maps task networks to sets of task networks. Informally, κ represents the actions of a critic, which finds problems in a task network and suggests a way to fix these problems. For example, suppose that we know that it is better to purchase both train tickets at the same time. Then $\kappa(N_4)$ might include the following task network, N_5:



Every critic function $\kappa()$ needs to satisfy two conditions: For any two task networks N_1 and N_2 such that $N_1 \in \kappa(N_2)$, any plan for N_1 should also be a plan for N_2, and whenever there is a plan for a task network N, $\kappa(N)$ should contain a task network for which there is a plan.

An *HTN planning domain* is a 4-tuple $\mathbf{D} = (O, M, \kappa, s_0)$, where O is a set of operators, M is a set of methods, κ is a critic function, s_0 is a state in $\mathcal{L}$ called the *initial state*, and N is a task network N representing some problem to be solved. The *language* of $\mathbf{D}$ is the planning language $\mathcal{L}$ generated by the constant symbols, predicate symbols, and task symbols in $\mathbf{D}$.

A *planning problem* is a pair $(\mathbf{D}, N)$, where $\mathbf{D}$ is a planning domain and N is a task network representing the problem to be solved. This generalizes the usual idea of an HTN planning problem, in which the problem to be solved would typically be represented by a single task.

Let $\mathbf{D} = (O, M, \kappa, s_0)$ be a planning domain, and $N = (G, C)$ be a task network, where $G = (T, \prec)$ is the partially ordered set of tasks and operators, and C is the set of constraints. Then red$(\mathbf{D}, N)$ is the set of all reductions of N. If every member of T is an operator, then we say that N is *primitive* (note that in this case, red$(\mathbf{D}, N) = \emptyset$. Now, suppose that N is primitive, and that

- $T' = (o_1, \ldots, o_n)$ is any total ordering of the operators of T that is consistent with $\prec$;
- θ is any substitution that assigns ground terms to all variables in T;
- the sequence of states $(s_0, s_1, \ldots, s_n)$ satisfies C, where $s_i = o\theta(s_{i-1})$ for $i = 1, \ldots, n$.

Then we say that the plan $P = T'\theta$ is a *completion* of N. We use compl$(N, \mathbf{D})$ to denote the set of all completions of N.

Let $\mathbf{D} = (O, M, \kappa, s_0)$ be a planning domain, and $N = (G, C)$ be a task network, where $G = (T, \prec)$ is the partially ordered set of tasks and operators, and C is the set of constraints. Consider the following recurrence:

$$S(\mathbf{D}, N) = \begin{cases} \text{compl}(N, \mathbf{D}) & \text{if } N \text{ is primitive;} \\ \bigcup_{N' \in \text{red}(\mathbf{D}, N)} S(\mathbf{D}, N') & \text{otherwise.} \end{cases}$$

What we would like to do is to define $S(\mathbf{D}, N)$ to be the set of all solutions to the planning problem $(\mathbf{D}, N)$. However, the problem with this is that S is ill-defined: depending on what κ and M are, there may be a number of functions S that satisfy Eq. (1). Let $\mathcal{S}$ be the set of all functions S that satisfy Eq. (1). Then we define the set of all *solutions* for the planning problem $(\mathbf{D}, N)$ to be the following set:

$$sol(\mathbf{D}, N) = \bigcap_{S \in \mathcal{S}} S(\mathbf{D}, N). \qquad (1)$$

It may not be immediately clear to the reader why this definition is correct. At the time that we submitted the original version of this paper for review, it was not completely clear to us either, and thus we were careful to state that this was strictly a preliminary definition. However, we have subsequently developed a semantics for HTN planning, which has allowed us to prove that $sol(\mathbf{D}, N)$ contains precisely those plans which we would want to consider to be solutions to $(\mathbf{D}, N)$. A discussion of that result is beyond the scope of this paper; readers are referred to (Erol *et al.*, 1993).

Two planning problems $(\mathbf{D}, N)$ and $(\mathbf{D}', N')$ are *equivalent* if they have the same solutions, i.e., if $sol(\mathbf{D}, N) = sol(\mathbf{D}', N')$.

An HTN planner is a procedure $\mathbf{P}$ which takes as input an HTN planning problem $(\mathbf{D}, N)$, and either exits with failure or returns a plan P. $\mathbf{P}$ is *sound* if whenever $\mathbf{P}(\mathbf{D}, N)$ returns an answer, that answer is in $sol(\mathbf{D}, N)$. $\mathbf{P}$ is *complete* if whenever $\mathbf{P}(\mathbf{D}, N)$ exits with failure, $sol(\mathbf{D}, N) = \emptyset$.

An Abstract Procedure for HTN Planning

Below, we present an abstract procedure for hierarchical nonlinear planning. This procedure, which is an instance of the function $\mathbf{P}$ defined earlier, is intended to be general enough to encompass most existing hierarchical nonlinear planning systems. As stated in

the theorems following the procedure, this procedure is sound and complete, and it has informedness properties analogous to those of the well-known A* search procedure (for further details and additional results, see (Erol *et al.*, 1993)).

procedure $\mathbf{P}_{HTN}(\mathcal{L}, A, M, \kappa, s_0, N_0)$:

1. Set $N := N_0$.
2. If N is primitive, then find a variable assignment and total ordering that satisfies the constraints in C_N, and halt with success. If such a variable assignment and ordering cannot be found, then exit with failure.
3. Choose a non-primitive task node t in N. (This is not a backtracking point.)
4. Nondeterministically choose a method $m = (t', N') \in M$ that matches t and a substitution θ that unifies t with t', and set $N := \text{reduce}(N, t, m, \theta)$.
5. Nondeterministically set $N :=$ some member of $\kappa(N)$.
6. Go to Step 2.

Theorem 1 (Soundness of $\mathbf{P}_{HTN}$) *Let* $\mathbf{D}$ *be any planning domain and* N_0 *be any planning problem in* $\mathbf{D}$. *If* $\mathbf{P}_{HTN}(\mathbf{D}, N_0)$ *returns an answer, then that answer is a plan in* $sol(\mathbf{D}, N_0)$.

Theorem 2 (Completeness of $\mathbf{P}_{HTN}$) *Whenever* $\mathbf{P}_{HTN}(\mathbf{D}, N)$ *exits with failure,* $sol(\mathbf{D}, N)$ *is empty.*

Theorem 3 (Informedness Theorem for Critics) *Let* $\mathbf{D}_1 = (O, M, \kappa_1, s_0)$ *and* $\mathbf{D}_2 = (O, M, \kappa_2, s_0)$ *be identical except for their critic functions. Suppose that for every task network* N,

- $\kappa_1(N) \subseteq \kappa_2(N)$;
- $sol(\mathbf{D}_2, N') = \emptyset$ *for every* $N' \in \kappa_2(N) - \kappa_1(N)$.

Then $\mathbf{D}_1$ *and* $\mathbf{D}_2$ *are equivalent, and the number of steps required by* $\mathbf{P}_{HTN}(\mathbf{D}_1, N) \leq$ *the number of steps required by* $\mathbf{P}_{HTN}(\mathbf{D}_2, N)$.

Theorem 4 (Informedness Theorem for Methods) *Let* $\mathbf{D}_1 = (O, M_1, \kappa, s_0)$ *and* $\mathbf{D}_2 = (O, M_2, \kappa, s_0)$ *be identical except for their sets of methods. Suppose that*

- $M_1 \subseteq M_2$;
- *for every task network* N, *task* t *in* N, *and method* m *in* $M_2 - M_1$ *that matches* t, $sol(\mathbf{D}_2, \text{reduce}(N, t, m(n))) = \emptyset$.

Then $\mathbf{D}_1$ *and* $\mathbf{D}_2$ *are equivalent, and the number of steps required by* $\mathbf{P}_{HTN}(\mathbf{D}_1, N) \leq$ *the number of steps required by* $\mathbf{P}_{HTN}(\mathbf{D}_2, N)$.

Conclusions

In this report, we have outlined a formal definition of HTN planning. Based on this definition, we have presented a nondeterministic HTN planning procedure $\mathbf{P}_{HTN}$ that is general enough to include most existing HTN planning procedures as special cases. We have presented theorems showing that $\mathbf{P}_{HTN}$ is sound and complete.

In addition, we have presented theorems showing that if $\mathbf{P}_{HTN}$ is run on planning problems P_1 and P_2 that equivalent except that P_2 has "more informed" methods and/or critics, then $\mathbf{P}_{HTN}$ will perform more efficiently on P_2. These results are analogous to the informedness results for the well-known A* search procedure.

While preliminary, these results give a hint of what can be done by developing a complete formalization of HTN planning. In (Erol *et al.*, 1993), we develop our formalization further, by presenting a semantics for HTN planning, and use this to present additional results. We hope to use this formalism to investigate topics such as the following:

- completeness, soundness, and complexity of various specific planning procedures;
- complexity and decidability of various classes of planning problems;
- relative expressivity of the HTN formalism vs. STRIPS-style formalisms;
- relative efficiency of various commitment strategies.

References

Chapman, David 1987. Planning for conjunctive goals. *Artificial Intelligence* 32:333–379.

Erol, K.; Hendler, J.; and Nau, D. S. 1993. Semantics for hierarchical task-network planning. Submitted for publication.

McAllester, David and Rosenblitt, David 1991. Systematic nonlinear planning. In *AAAI-91*. 634–639.

Minton, S.; Bresna, J.; and Drummond, M. 1991. Commitment strategies in planning. In *Proc. IJCAI-91*.

Sacerdoti, Earl D. 1990. The nonlinear nature of plans. In Allen, James; Hendler, James; and Tate, Austin, editors 1990, *Readings in Planning*. Morgan Kaufman. 206–214. Originally appeared in *Proc. IJCAI-75*.

Tate, Austin 1977. Generating project networks. In *Proc. 5th International Joint Conf. Artificial Intelligence*.

Wilkins, David E. 1984. Domain-independent planning: Representation and plan generation. *Artificial Intelligence* 22(3).