On the Feasibility of Planning Graph Style Heuristics for HTN Planning

Ron Alford
University of Maryland
College Park, MD, USA
ronwalf@volus.net

Vikas Shivashankar
University of Maryland
College Park, MD, USA
svikas@cs.umd.edu

Ugur Kuter
SIFT, LLC
Minneapolis, MN, USA
ukuter@sift.net

Dana Nau
University of Maryland
College Park, MD, USA
nau@cs.umd.edu

Abstract

In classical planning, the polynomial-time computability of propositional delete-free planning (planning with only positive effects and preconditions) led to the highly successful Relaxed Graphplan heuristic. We present a hierarchy of new computational complexity results for different classes of propositional delete-free HTN planning, with two main results:

We prove that finding a plan for the delete-relaxation of a propositional HTN problem is NP-complete: hence unless P=NP, there is no directly analogous GraphPlan heuristic for HTN planning. However, a further relaxation of HTN planning (delete-free HTN planning with task insertion) is polynomial-time computable. Thus, there may be a possibility of using this or other relaxations to develop search heuristics for HTN planning.

1 Introduction

Planning has been shown to be theoretically intractable in general. Bylander (1994) showed that even the simplest interesting variant of classical planning is PSPACE-complete. Hierarchical Task Network (HTN) planning is even harder: depending on the particular variant, the complexity can be anywhere from EXPTIME to undecidable (Erol, Hendler, and Nau 1996).

To combat the complexity of classical planning, modern classical planners use efficiently computable state-based heuristics that often work very well in practice (Helmert 2006; Hoffmann and Nebel 2001; Bonet and Geffner 2001; Nguyen and Kambhampati 2001). The most influential among these is arguably the *Relaxed Planning Graph* heuristic used in the FF planner (Hoffmann and Nebel 2001), which solves the propositional delete-free version of the given problem in polynomial time, and computes a heuristic value based on that solution. Relaxed planning-graph heuristics have since been developed for a variety of purposes, e.g., probabilistic planning (Yoon, Fern, and Givan 2007; Teichteil-Königsbuch, Kuter, and Infantes 2010), propositional landmark generation (Richter and Westphal 2010), metric planning (Hoffmann 2003).

In this paper, for propositional delete-free HTN planning, we prove results about the complexity of two well-known decision problems, plan-existence and k-length-plan-existence, under various conditions.

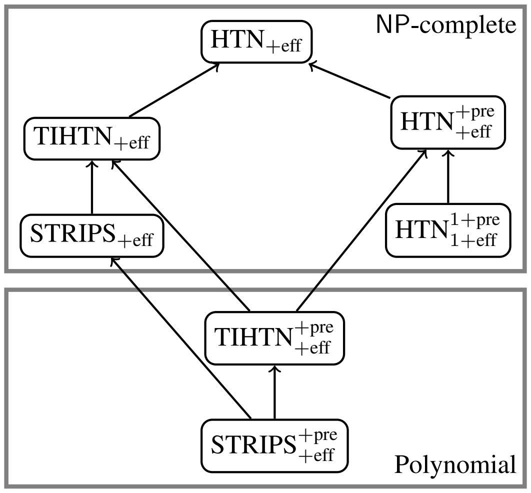
NP-complete
$\text{HTN}_{+\text{eff}}$
$\text{TIHTN}_{+\text{eff}}$
$\text{HTN}^{+\text{pre}}_{+\text{eff}}$
$\text{STRIPS}_{+\text{eff}}$
$\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$
$\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$
$\text{STRIPS}^{+\text{pre}}_{+\text{eff}}$
Polynomial

Figure 1: Complexity of plan-existence for propositional delete-free STRIPS and HTN planning with various restrictions (k-length-plan-existence is NP-complete in all cases). Arrows represent subclass relationships. The STRIPS results are from (Bylander 1994); the other results are new.

Fig. 1 summarizes the results, using the following notation. TIHTN is propositional HTN planning with *task insertion* (see Section 3 and (Geier and Bercher 2011)); "+pre" (resp. "+eff") means all preconditions (resp. effects) are positive; "1+pre" (resp. "1+eff") means at most one positive and no negative preconditions (resp. effects). Here is how the results bear on the feasibility of relaxation-based search heuristics for HTN planning:

- Even for very restricted cases, delete-free propositional HTN planning is NP-complete. Thus unless P=NP, there is no direct analogy of Relaxed GraphPlan for HTN problems.
- If the HTN planning semantics is modified to allow task insertion and all of the preconditions and effects are positive, then plan-existence is polynomial-time computable. Thus, it may be possible to use this or other relaxations to develop search heuristics for HTN planning.

2 Basics

In this section, we present a propositional HTN planning formalism, using the notation presented in (Geier and Bercher 2011).

It will be important for us to have a notation for the *restriction* of a function or relation to some subset of its domain. For this, we will use a *bar notation* that is defined as follows. For a binary relation $R \subseteq A \times A$, the restriction of R to any $X \subseteq A$ is

$$R|_X = \{(p_1, p_2) \in R \mid p_1, p_2 \in X\}.$$

Similarly, for a function $f : P \to Q$, the restriction of f to any $X \subseteq P$ is

$$f|_X = \{f(p) = q \mid p \in X\}.$$

Given a set of task names X, a *task network* is a tuple $tn = (T, \prec, \alpha)$ such that:

- T is a finite nonempty set of *task symbols*.
- $\prec$ is a partial order over T.
- $\alpha : T \to X$ is a mapping from the task symbols to a finite set of task names.

The task symbols function as place holders for task names, allowing multiple instances of a task name to exist in a task network (Erol, Hendler, and Nau 1994). We refer to the set of all task networks over a set of task names X as TN_X. An *HTN domain* is a tuple (L, C, O, M), where L is a finite set of proposition symbols, C is a finite set of compound task names, O is a finite set of primitive task names, with $O \cap C = \emptyset$, and $M \subseteq C \times TN_{C \cup O}$ is a set of methods over C and O.

Each primitive task name $o \in O$ denotes a *planning operator*: $(prec(o), add(o), del(o))$. $prec(o)$ is a propositional formula over L, and $add(o)$ and $del(o)$ are disjoint subsets of L. Note that the semantic models for the operators in O forms an implicit *state transition function* for the planning domain:

$$\gamma : 2^L \times O \to 2^L,$$

where:

- A state is any subset of L. The finite set of states in a planning domain is denoted as 2^L in the above definition of γ;
- $\gamma(s, o)$ is defined iff $s \models prec(o)$; and
- $\gamma(s, o) = (s \setminus del(o)) \cup add(o)$.

We call a task network *primitive* if $\alpha(t) \in O$ for every $t \in T$. Otherwise, the task network is *non-primitive*.

We can *decompose* a non-primitive task network $tn_1 = (T_1, \prec_1, \alpha_1)$ if there is a non-primitive task $t \in T_1$ such that $\alpha(t) \in C$ and has a corresponding method $m = (\alpha(t), (T_m, \prec_m, \alpha_m)) \in M$. More formally, we define the notion of *task decomposition* as follows. Assume without loss of generality that $T_1 \cap T_m = \emptyset$. Then the decomposition of tn_1 by m into a task network tn_2 (written $tn_1 \xrightarrow{t,m}_D tn_2$) is given by:

$$\begin{aligned}
T_1' &:= T_1 \setminus \{t\}\,; \\
T_2 &:= T_1' \cup T_m; \\
\prec_2 &:= \prec_1 |_{T_1'} \\
&\quad \cup \prec_m \\
&\quad \cup \{(t_1, t_2) \in T_1' \times T_m \mid (t_1, t) \in \prec_1\} \\
&\quad \cup \{(t_2, t_1) \in T_m \times T_1' \mid (t, t_1) \in \prec_1\}\,; \\
\alpha_2 &:= \alpha_1|_{T_1'} \cup \alpha_m; \\
tn_2 &:= (T_2, \prec_2, \alpha_2)\,.
\end{aligned}$$

If there is a finite sequence of task decompositions from $tn_1 \to_D tn_2 \to_D \ldots \to_D tn_n$, then we write $tn_1 \to_D^* tn_n$.

An *HTN planning problem* is a tuple (D, s_0, tn_0), where $D = (L, C, O, M)$ is an HTN domain, $s_0 \in 2^L$ is a state in D, and $tn_0 = (\{t_0\}, \emptyset, \{(t_0, x_0)\})$ is the initial task network containing a single task x_0.

A task network tn is *executable* in a state s_0 for domain D if tn is primitive and there exists some total ordering (consistent with $\prec$ in tn) over the tasks $t_1, \ldots, t_n$ and the sequence of states $s_1, \ldots, s_n$ that arise from applying the primitive tasks (i.e., actions) $t_1, \ldots, t_n$ in that order in the initial state s_0: i.e.,

$$\forall_{i=0\ldots n-1} \gamma(s_i, \alpha(t_{i+1})) = s_{i+1}$$

We say that tn^* is an *HTN solution* to a planning problem $P = (D, s_0, tn_0)$ if tn^* is executable in s_0 and $tn_0 \to_D^* tn^*$. In this paper, we are concerned with two HTN decision problems: *plan-existence*, for whether a problem has any solution, and *k-length-plan-existence*, for whether a problem has a solution of k or fewer operators.

For the purposes of this paper, we consider only *delete-free* planning problems, where operators contain only positive effects (i.e., $\forall_{o \in O} del(o) = \emptyset$). Deferring to Bylander, we refer to this restricted class of problems as members of $\text{HTN}_{+\text{eff}}$. When problems are further restricted to contain only operators with positive preconditions, we say these problems belong to $\text{HTN}^{+\text{pre}}_{+\text{eff}}$. In the highly restricted case where both the preconditions and effects of operators contain at most a single positive literal, we say these problems belong to $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$.

3 Delete-Free Task Insertion HTN Problems

Before we get to delete-free HTN problems, we shall first consider delete-free versions of a variant of HTN planning: HTN Planning with Task Insertion (TIHTN) (Geier and Bercher 2011). In TIHTNs, a problem is still modeled in terms of an initial state and a task network that needs to be decomposed, but insertion of primitive tasks is now allowed without requiring them to be inserted by the decomposition of a compound task that is present in the task network.

Formally, given a primitive task o and a task network $tn = (T, \prec, \alpha)$, we can insert o into tn to obtain a new task network tn' by generating a new symbol t and letting $tn' = (T \cup \{t\}, \prec, \alpha \cup (t, o))$. We write $tn \to_I^* tn'$ if tn' can be obtained from tn by any sequence of task insertions.

A TIHTN planning problem is syntactically identical to an HTN planning problem. A TIHTN planning problem (D, s_0, tn_0) has a solution if:

- tn_0 is primitive and executable in s_0.
- $tn_0 \rightarrow_I tn'$ and (D, s_0, tn') has a solution
- $tn_0 \rightarrow_D tn'$ and (D, s_0, tn') has a solution

Note that task insertion and decomposition commute, and so we can always reorder task insertions to come before decompositions. For instance, if the TIHTN problem (D, s_0, tn_0) is solvable, there are task networks tn_1, tn_1', and tn_2 such that $tn_0 \rightarrow_I^* tn_1 \rightarrow_D^* tn_2$, $tn_0 \rightarrow_D^* tn_1' \rightarrow_I^* tn_2$, and tn_2 is executable in s_0.

As Geier and Bercher show, TIHTN planning relaxes HTN planning enough to regain decidability of plan existence even in cases when the original HTN problem remains undecidable.

We show in the following theorem that plan-existence for TIHTN problems with positive preconditions and effects ($\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$) is polynomial-time computable.

Theorem 3.1 *If $P = (D, s_0, tn_0)$ (where $D = (L, C, O, M)$) is a Task Insertion HTN planning problem with positive preconditions and effects ($TIHTN^{+pre}_{+eff}$), then plan-existence for P is decidable in time $O\left(|O|^2 + |M|^2\right)$.*

Proof.

Iteratively insert and apply operators from O to s_0 until we reach the fixed point state s where no new operators are applicable, much like Relaxed GraphPlan (taking $O\left(|O|^2\right)$ time).

Given positive preconditions and effects, no operator application can make another operator inapplicable (or inapplicable) from the fixed point state, and for every state s' obtainable from s_0, if $s' \models prec(o)$ then $s \models prec(o)$. Thus, we can perform a bottom-up parse of the methods to show which non-primitive tasks are executable in s:

The following algorithm iterates through the list of methods at most $|M|$ times finding a solution for at least one non-primitive task in all but its last iteration, starting from the non-primitive tasks.

1. For every primitive task $o \in O$ where $s \models prec(o)$, mark o as solvable.
2. Iterate through the methods in M. If $m = (c, tn)$ is a method such that all the tasks names in tn are marked as solvable, mark c as solvable.
3. Repeat line 2 if it marked any new task names as solvable.
4. Return $TRUE$ if all task names in tn_0 are solvable, return $FALSE$ otherwise.

Since at least one method is marked in every pass, this takes $O\left(|O| + |M|^2\right)$ time, resulting in an overall time complexity of $O\left(|O|^2 + |M|^2\right)$.

□

Table 1: Summary of results from Section 3.

Problem	plan-existence	k-length-plan-existence
$\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$	P	NP-hard
$\text{TIHTN}_{+\text{eff}}$	NP-hard	NP-hard
$\text{HTN}^{+\text{pre}}_{+\text{eff}}$	-	NP-hard
$\text{HTN}_{+\text{eff}}$	NP-hard	NP-hard

We shall now establish lower bounds on complexities of both plan-existence and k-length-plan-existence for the remaining delete-free TIHTN planning classes.

Firstly, we note that (delete-free) TIHTN problems can be encoded as (delete-free) HTN problems as follows: given a TIHTN domain $D = (L, C, O, M)$, we add for every $t \in C$ and $o \in O$ a method to M that decomposes t into a pair of subtasks $\langle o, t \rangle$. Similarly, we can also show that (delete-free) STRIPS problems can be encoded as (delete-free) TIHTN problems by simply adding a dummy operator o with the goal as its precondition and no effects and letting the initial task network consist of o.

Since we know that plan-existence for $\text{STRIPS}_{+\text{eff}}$ and k-length-plan-existence for both $\text{STRIPS}^{+\text{pre}}_{+\text{eff}}$ and $\text{STRIPS}_{+\text{eff}}$ is NP-hard (Bylander 1994), it follows immediately from the encoding from STRIPS to TIHTN problems that plan-existence for $\text{TIHTN}_{+\text{eff}}$ and k-length-plan-existence for both $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ and $\text{TIHTN}_{+\text{eff}}$ are also NP-hard.

Now using the encoding from TIHTN to HTN problems, we can similarly lower bound the complexities for some HTN planning problem classes. In particular, we can show that plan-existence for $\text{HTN}_{+\text{eff}}$ is NP-hard and that k-length-plan-existence for both $\text{HTN}_{+\text{eff}}$ and $\text{HTN}^{+\text{pre}}_{+\text{eff}}$ are NP-hard.

Table 1 summarizes the complexity results from this section. One thing yet to be done is to estimate the complexity of solving $\text{HTN}^{+\text{pre}}_{+\text{eff}}$problems. As we shall see in the following section, while the task insertion variant of this problem ($\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$) is solvable in polynomial time, $\text{HTN}^{+\text{pre}}_{+\text{eff}}$problems are much harder to solve.

4 Solving $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$ Problems is NP-hard

We begin our analysis on delete-free HTN planning problems by focusing on a restricted case where operators have only one positive effect and one positive precondition, which we refer to as $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$. In the following theorem, we establish that plan-existence is NP-hard for $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$ problems (and thus NP-hard for $\text{HTN}^{+\text{pre}}_{+\text{eff}}$) even when:

- Every method is totally ordered
- Every method is *regular*, such that non-primitive tasks only occur as the last task in the method.
- The methods are *acyclic*, meaning there are only a finite number of solutions to the initial problem.

Theorem 4.1 *Plan existence for HTN^{1+pre}_{1+eff} planning is* NP-*hard.*

Proof. Let $E = e_1 \wedge e_2 \wedge \ldots \wedge e_n$ be a CNF-SAT formula, where each conjunct is a disjunction over a set of variables $v_1, \ldots, v_m$ and their negations.

To give an encoding, we need to construct a delete-free HTN planning problem where any solution implies a satisfying assignment for E, and no solution implies E is unsatisfiable. The encoding of E is the HTN domain $D = (L, C, O, M)$ and problem $(D, \emptyset, tn_0)$, all given below.

Let the set of propositions L consist of two propositions for each variable, $v_i\text{-}true$ and $v_i\text{-}false$ representing a true and false assignment to v_i, respectively.

Let the set of operators O consist of four operators for each variable v_i, two for setting the value of the variable and two for checking its truth or negation:

- An operator $set\text{-}v_i\text{-}true$, with

$$\begin{aligned} prec(set\text{-}v_i\text{-}true) &= true, \\ add(set\text{-}v_i\text{-}true) &= \{v_i\text{-}true\}, \\ del(set\text{-}v_i\text{-}true) &= \emptyset. \end{aligned}$$

- An operator $set\text{-}v_i\text{-}false$, with

$$\begin{aligned} prec(set\text{-}v_i\text{-}false) &= true, \\ add(set\text{-}v_i\text{-}false) &= \{v_i\text{-}false\}, \\ del(set\text{-}v_i\text{-}false) &= \emptyset. \end{aligned}$$

- An operator $check\text{-}v_i\text{-}true$, with

$$\begin{aligned} prec(check\text{-}v_i\text{-}true) &= v_i\text{-}true, \\ add(check\text{-}v_i\text{-}true) &= del(check\text{-}v_i\text{-}true) = \emptyset. \end{aligned}$$

- An operator $check\text{-}v_i\text{-}false$, with

$$\begin{aligned} prec(check\text{-}v_i\text{-}false) &= v_i\text{-}false, \\ add(check\text{-}v_i\text{-}false) &= del(check\text{-}v_i\text{-}false) = \emptyset. \end{aligned}$$

The set of non-primitive tasks C consist of the tasks (1) $set\text{-}v_i$ for each $i \leq m$, which chooses an assignment for the variable v_i, and (2) $check\text{-}e_j$ for each $j \leq n$, which checks whether the conjunct e_j in the given CNF-SAT formula E is satisfied with the current variable assignments. We shall now describe the methods M that decompose tasks in C.

For the task $set\text{-}v_i$, we introduce two methods: one which calls $set\text{-}v_i\text{-}true$ and then $set\text{-}v_{i+1}$, and another which calls $set\text{-}v_i\text{-}false$ and then $set\text{-}v_{i+1}$. For $set\text{-}v_m$, the task corresponding to the last variable v_m, we introduce two methods as above, one each for setting v_m as `true` and `false` respectively; instead of calling $set\text{-}v_{i+1}$ however (as in the earlier cases), both these methods will now initiate checking satisfiability of the conjuncts in E by calling $check\text{-}e_1$.

Let us now construct methods for $check\text{-}e_i$. Since e_i is a disjunction of literals, let $l_1, \ldots, l_k$ be the disjuncts of e_i. We shall now write a method for each literal in e_i; since e_i is a disjunction, it suffices if one of these methods succeeds in decomposing $check\text{-}e_i$. In particular, for each literal l_j, we encode a method for $check\text{-}e_i$: if l_j is of the form $\neg v_l$ for some variable v_l, then the method calls $check\text{-}v_l\text{-}false$ followed by $check\text{-}e_{i+1}$. Otherwise, l_j is of the form v_l, and the method calls $check\text{-}v_l\text{-}true$ followed by $check\text{-}e_{i+1}$. The methods for $check\text{-}e_m$ omit the call to check the next expression.

The initial task network tn_0 contains a single task, $set\text{-}v_1$. Any primitive decomposition of tn_0 must first call $set\text{-}v_i\text{-}true$ or $set\text{-}v_i\text{-}false$ (but not both) for each variable, and then check that one literal is true for each conjunct in E. Thus there exists a solution to the HTN problem iff there is a satisfying assignment for the variables in E.

Since the encoding is linear with respect to the length of E and CNF-SAT is NP-hard, delete-free HTN planning is NP-hard. □

Any of the three restrictions on method structure mentioned at the start of this section is enough to place a HTN planning problem in a decidable fragment of the language (Erol, Hendler, and Nau 1996). This leaves only two obvious syntactic restrictions that would make a delete-free HTN problem solvable in polynomial time without relaxing the semantics: either restrict the initial task network to be primitive, or restrict all operators to have zero effects. Thus, we can safely say that delete-free HTN planning, except in the most trivial cases, is NP-hard.

5 Showing HTN$_{+\text{eff}}$ Problems are in NP

Here we show that if there is a solution of length k to a delete-free HTN planning problem, then there exists a polynomial size witness, verifiable in polynomial time, proving that there exists a solution of size k or smaller. This places both plan-existence and k-length-plan-existence in NP for delete-free HTN planning.

The outline of the proof is as follows: We present *decomposition trees* (Geier and Bercher 2011), which can be used as a witness that a task network is derivable from the initial network, and these trees can be verified in time polynomial in the size of the tree. We then digress to show that deciding whether a problem has a solution when the primitive tasks that change the state are fixed in advance is in NP. Since solutions in delete-free domains can only change the state a polynomial number of times, this lets us use a decomposition tree of polynomial width as part of the witness to the solvability of HTN$_{+\text{eff}}$ problems. Finally, we also provide a polynomial bound on the height of a decomposition tree necessary to show that problem is solvable.

Decomposition Trees

Geier et al (Geier and Bercher 2011) introduced the idea of *decomposition trees*, which is a representation of how the initial compound task c_I can be transformed to a task network tn via a sequence of decompositions.[1] We present their definitions below, modified slightly to suit our purposes.

[1]Note that the restriction for having a single task for the initial task network of an HTN planning problem is only for the sake of simplifying the exposure of our theoretical results; our definitions and theorems can be adapted to work without this restriction by generalizing the notion of decomposition trees, described below, to decomposition forests.

Given a planning problem P, a *decomposition tree* $g = (T, E, \prec, \alpha, \beta)$ is a five-tuple satisfying the following properties:

- (T, E) is a tree with nodes T and directed edges E pointing towards the leaves;
- $\prec$ is a partial order defined over T;
- $\alpha : T \to C \cup O$ is a labeling function that labels the nodes in T with task names;
- β is a labeling function that labels each inner node with a method $m = (c, tn_m)$ and an isomorphism from tn_m to the children of that node.

Moreover, we define $T(g)$ to refer to the tasks of g and $\mathsf{ch}(g, t)$ to refer to the direct children of $t \in T(g)$ in g.

The following definition states the conditions under which a decomposition tree encodes a decomposition of the initial task network. A decomposition tree $g = (T, E, \prec, \alpha, \beta)$ is *valid* with respect to a planning problem $P = (D, s_0, c_I)$ if and only if the root node of g is labeled with the initial task name c_I and for any inner node t, where $\beta(t) = ((c, tn_m), f)$, the following conditions hold:

1. $\alpha(t) = c$,
2. f is a valid isomorphism of the task network induced in g by $\mathsf{ch}(g, t)$ and tn_m; i.e.
$$(\mathsf{ch}(g,t), \prec|_{\mathsf{ch}(g,t)}, \alpha|_{\mathsf{ch}(g,t)}) \underset{f}{\cong} tn_m,$$
3. $\forall t' \in T, c' \in \mathsf{ch}(g, t)$, it holds that
   (a) if $t \prec t'$ then $c' \prec t'$;
   (b) if $t' \prec t$ then $t' \prec c'$.
4. there are no other ordering constraints in $\prec$ other than those demanded by conditions 2 and 3.

Informally, the above conditions capture the following checks for each inner node t: condition 1 verifies the applicability of the method $m = \beta(t)$ that t is labeled with; condition 2 verifies that m's task network is correctly represented in the tree; condition 3 ensures that the ordering constraints are inherited correctly after the application of m; and condition 4 ensures the minimality of $\prec$.

The definition of a decomposition tree and its validity to an HTN planning problem is identical to Geier and Bercher's definition, save for the addition of the explicit isomorphism at each inner node t, mapping $\mathsf{ch}(g, t)$ to the subtask network of the method applied at t. This modification is made so that the validity of a decomposition tree can be checked in time polynomial in the size of the tree[2]. We would also like to point out that the theoretical results in (Geier and Bercher 2011) still hold unchanged even with these modifications. This is an important point as we shall be using their theorems (which they proved under their definition) in our proofs.

Note that the leaves of a decomposition tree g form a task network, which is called the yield of g. Formally, the *yield of a decomposition tree* $g = (T, E, \prec, \alpha, \beta)$ is a task network defined as follows. Let $T' \subseteq T$ be the set of all leaf nodes in g. Then, $\mathsf{yield}(g) = (T', \prec|_{T'}, \alpha|_{T'})$.

[2]Since graph isomorphism is not known to be in P, this would not be possible without our modification.

Geier and Bercher (2011) use the above definitions to prove the following useful property of valid decomposition trees:

Theorem 5.1 *Given a planning problem $P = (D, s_0, c_I)$, the following holds for any task network $tn \in TN_{C\cup O}$. There exists a valid decomposition tree g with* $\mathsf{yield}(g) = tn$ *if and only if $c_I \to_D^* tn$.*

In other words, the reachability of tn from c_I via a sequence of method decompositions can be proved by providing a valid decomposition tree for the problem P whose yield is tn. This property, as we shall see later, will be instrumental in proving that delete-free HTN planning is in NP.

Given a decomposition tree $g = (T, E, \prec, \alpha, \beta)$ and a node $t \in T$, the subtree of g induced by t, written as $g[t]$, is
$$g[t] = (T', E', \prec|_{T'}, \alpha|_{T'}, \beta|_{T'}),$$
where (T', E') is the subtree of (T, E) rooted at t.

Definition 5.2 *Let $g = (T, E, \prec, \alpha, \beta)$ be a decomposition tree and $t_i, t_j \in T$ be two nodes of g. The result of the subtree substitution of t_i with t_j on g, written as $g[t_i \leftarrow t_j]$, is given as follows:*

- *If t_i is the root node of g, then $g[t_i \leftarrow t_j] = g[t_j]$.*
- *Otherwise, $g[t_i \leftarrow t_j] = (T', E', \prec|_{T'}, \alpha|_{T'}, \beta|_{T'})$, with*
  - $T' = (T \setminus T(g[t_i])) \cup T(g[t_j])$,
  - $E' = E|_{T'} \cup \{(p, t_j)\}$*, where p is the parent node of t_i in g.*

Note that this operation in general will *not* lead to valid decomposition trees. However, if applied under the right conditions, the result of the subtree substitution can still describe valid decompositions as described by the following result (Geier and Bercher 2011):

Theorem 5.3 *Let $g = (T, E, \prec, \alpha, \beta)$ be a valid decomposition tree for an HTN planning problem P. If we are given two nodes $t_i \in T, t_j \in T(g[t_i])$ such that $\alpha(t_i) = \alpha(t_j)$, then $g[t_i \leftarrow t_j]$ is also a valid decomposition tree for P.*

In other words, if t_i and t_j map to the same task names and t_j is a descendant of t_i in g, then replacing t_i (and its subtree) with t_j (and its subtree) still results in a valid decomposition tree. This technique can therefore be used to eliminate cyclic decompositions from a tree while still retaining validity.

Forming a witness to the solvability of an HTN problem

We are going to use decomposition trees to show that delete-free HTN planning is in NP. Note that if a valid decomposition tree's yield is primitive and executable, then we can use the tree as a checkable proof that its problem is solvable. However, even in the restricted case where none of the operators have an effect, the minimal solution size (measured in the number of tasks) may still be exponential. So we need to be able to present a witness that includes both a tree with a non-primitive yield, and a polynomial size proof that some expansion of that yield is executable.

Definition 5.4 *Let $P = (D, s_0, tn_0)$ be an HTN planning problem, where $D = (L, C, O, M)$ and $tn_0 = (T_0, \prec_0, \alpha_0)$. A* state-transition preserving *solution for P is one in which the only state-changing actions are the ones that were already in tn_0, i.e., it is a primitive task network tn such that:*

- *$tn_0 \rightarrow^*_D tn$, where $tn = (T, \prec, \alpha)$*
- *tn has an executable ordering over its tasks $(t_1, \ldots, t_n$, executing over the states s_0 to $s_n)$*
- *If $t_i \notin T_0$ then $s_{i-1} = \gamma(s_{i-1}, \alpha(t_i)) = s_i$*

Given a sequence of states, a *solution table* for finding a state-transition preserving solution consists of a row for each combination of start state, end state, and task name. Each row in a solution table has a *value*, defined as follows:

- For each row with a primitive task name, the value of that row is 1 if the ground instance of the operator for that primitive task name is both applicable in any state between the start state and end state, inclusively, and the operator does not change said state. Otherwise, the value of the row is ∞.
- For each row with a non-primitive task name, we associate a method used to decompose the task, and a set of pointers back into the table supporting that the method is executable (without changing the state) between the start and end state for the row. The value for the row is then the sum of the values of its supporting rows.

We can check the table by first checking the primitive entries of the table, and then repeatedly scanning the table to find rows whose supports have already been checked. This leads into the following lemma:

Lemma 5.5 *Both the plan-existence and the k-length-plan-existence problems for finding a state-transition preserving solution are in* NP.

Proof. Let $P = (D, s_0, tn_0)$ be an HTN planning problem where $D = (L, C, O, M)$, $tn_0 = (T_0, \prec_0, \alpha_0)$ such that P has a state-transition preserving solution.

By definition, in any state-transition preserving solution, only the primitive tasks already in tn_0 may change the state. So given a fixed, executable ordering over the primitive tasks of tn_0 and the states associated with that ordering $(s_0, \ldots, s_n)$, the decompositions of non-primitive tasks in tn_0 interact with each other only in what states they start and end on (constraining the end and start states, respectively, of tasks required to come before or after). Start and end states for a task determine what decompositions (if any) are executable over that sequence of states. This lets us construct a solution table as described above.

Once the solution table is constructed, a witness to the solvability of P (i.e., a witness that there exists a state-preserving solution for P) consists of a total order over the primitive tasks of tn_0, a solution table described above for the sequence of states traversed by those primitive tasks, and a set of pointers into the table for each non-primitive task in tn_0. The value of the solution is the sum of the primitive tasks in the row of the solution table that holds tn_0, plus the sum of the values sizes of the supporting table entries. Since the validity of the ordering and table are verifiable in polynomial time, both plan-existence and k-length-plan-existence for finding a state-transition preserving solution are both in NP. ☐

We can now use a decomposition tree as a proof that an HTN problem is solvable, even if the yield of that tree is non-primitive:

Definition 5.6 *Let tn be a task network, g be valid decomposition tree of tn, and stp be a witness that the yield of g has a state-transition preserving solution. Then, a* witness *to the solvability of an HTN problem $P = (D, s, tn)$ is the pair (g, stp) of a valid decomposition tree g of tn with stp.*

Since checking the validity of a tree is polynomial in the size of the tree, and checking the witness that the yield of the tree has a state-transition preserving solution is polynomial in the size of the yield and the number of task names, it follows that the combined witness is also in P. Furthermore, note that every solvable HTN planning problem has a witness, even non-delete-free problems. However, the existence of a polynomial-sized witness is only likely in delete-free planning, where a fix-point state is reachable in a polynomial number of actions. In the remaining sections, we show that delete-free HTN planning problems always have a witness of polynomial size.

Bounding the breadth of the witness tree

Given a delete-free HTN problem and its witness, (g, stp), we know there are at most $|O|$ primitive tasks which change the state in any execution of the yield of g, where O is the set of operators. We now show how to restrict a decomposition tree to its minimal valid subtree that contains those operators.

Definition 5.7 (Saplings) *Given a tree $g = (T, E, \prec, \alpha, \beta)$ and a set of tree nodes $S \subset T$, let T' be the set of nodes along any path from a node in S to the root of g (inclusively) and the siblings of each and every node along the path. Formally, T' is the smallest subset of T such that:*

- *$S \subseteq T'$*
- *$\forall_{t,t' \in T} t' \in T' \land (t, t') \in E \implies t \in T'$*
- *$\forall_{t,t_1,t_2 \in T'} (t_1 \in T') \land \{(t, t_1), (t, t_2)\} \subseteq E \implies t_2 \in T'$*

Then the sibling-augmented path tree *or S-sapling of T is $\left(T', E_{|T'}, \prec_{|T'}, \alpha_{|T'}, \beta_{|T''}\right)$, where T'' contains the inner nodes of T'.*

Proposition 5.8 *Given a tree $g = (T, \prec, \alpha, \beta)$ and a set $S \subseteq T$, then the S-sapling of g is a valid decomposition tree.*

Proof. Any subtree of g containing the root satisfies all but condition 3 of definition a valid decomposition tree. Since the construction of a sapling either preserves all children of node or none of them, condition 3 also holds. ☐

Given a witness (g, stp) for a delete-free problem, we can create a sapling using just the primitive tasks that change the state.

Lemma 5.9 *Let (g, stp) be a witness that a delete-free HTN problem $P = (D, s_0, tn_0)$ with domain $D = (L, C, O, M)$ is solvable. Let $(T, \prec, \alpha) = yield(g)$, and let $S \subseteq T$ be the set of tasks that change the state in the order specified by stp. Then if g' is the S-sapling of g, there exists a witness stp' such that the yield of g' has state-transition preserving solution of the same size or smaller than the yield of g.*

Proof. Given that stp is the witness that g has a state-transition preserving solution, let $(<, B, R) = stp$, where:

- $<$ is $\langle t_1, \dots, t_n \rangle$ which is the total ordering over the primitive tasks in the yield of g. Let $s_1, \dots, s_m$ be the distinct states that sequence produces (omitting repeated states).
- B is the solution table for the sequence $s_1, \dots, s_m$. Assume WLOG that B is optimal, giving the smallest possible solution size for each entry in B.
- R is the set of pointers into B for non-primitive tasks in the yield of g.

Let S be the set of primitive tasks that change the state ($\{t_i \in yield(g) \mid s_{i-1} \neq s_i\}$), and let g' be the S-sapling of g.

Now we provide a witness that g' has a state-transition preserving solution. Let $<'$ be the same ordering as $<$ restricted to tasks in S. Since tasks in $<$ but not in $<'$ did not change the state, an execution of $<'$ produces the same sequence $s_1, \dots, s_m$ of distinct states that $<$ did, and so we can reuse the same solution table B.

For the set of supports, any task t in the yield of g' which was not in the yield of g must have children in the yield of g which, under the given ordering $<$, were all either primitive tasks which did not change the state or were non-primitive with state-transition preserving expansions with entries in R. So B, the solution table, must have an entry for s_i, s_j, t with finite value, where s_i and s_j are the first state and last in the sequence $s_0, \dots, s_m$ where either primitive descendant was executed or the first state used in R for a non-primitive descendant. So we can construct a new set of supports R' using the above method for any task in the yield of g' but not in R, and directly using the entry from R otherwise.

So $stp' = (<', B, R')$ is a witness that g' has a state-transition preserving solution. Moreover, since B remains the same and R' was calculated from B and R, stp' must indicate that g' has a solution in B with the same or lower value as g. □

Bounding the height of the witness tree

The above lemma lets us take any witness (g, stp) to a problem's solvability and construct a new witness which is composed of a polynomial number of paths to the root g (plus siblings). This is not quite enough to show that delete-free planning is in NP, since those paths may not be polynomial in length. However, in those cases, we can use a variant of the pumping lemma (Comon et al. 2007) to produce a new witness with polynomially-bounded length paths:

Theorem 5.10 *Let $P = (D, s_0, tn_0)$ (where $D = (L, C, O, M)$) be a solvable delete-free HTN planning problem, with P having a minimal solution size of k . Then there exists a witness (g, stp) that P has a solution size of $\leq k$, with $|T(g)| \leq m \cdot |C| \cdot |O|^2$, where m is the size of the largest task network in M.*

Proof. If P is solvable, there exists a tree g_p with an executable, primitive yield of optimal size k. Let (g, stp) be the S-sapling witness as constructed above in lemma 5.9, where S is the set of tasks in the witness that change the state. Then (g, stp) is a witness that P has a solution of size $\leq k$.

Suppose g has a height that is greater than $|C| \cdot |O|$. Since g is constructed from a series of paths from nodes to the root, this means that there is some path from a node in S to the root of that length.

Let $t_1, \dots, t_n$ be the tasks along that path. Since that path is joined at most $|S| - 1$ times by other paths from S to the root ($|S| \leq |O|$) and since there are only $|C|$ task names to assign, there must be some segment $t_i, \dots, t_j$ between joins such that $\alpha(t_i) = \alpha(t_j)$, and no descendants of t_i not on the path to t_j has a descendants that is in S.

Since no descendants of t_i that are not also a descendants of t_j are in S, then all of those descendants must have a state-transition preserving solution under stp'. Let $g' = g\,[t_i \leftarrow t_j]$ be the tree obtained by substituting t_j for t_i. Since we only removed tasks which did not change the state, the yield of g' is a strict subset of the yield of g. So we can create a witness stp' that g' has a state-transition preserving solution by restricting the set of supports in stp to the tasks remaining in the yield of g'. That solution must have a size strictly less than k. This would violate our assumption k was the minimal solution size.

So g must have a height that is less than or equal to $|C| \cdot |O|$. □

Since we can always find a polynomial sized witness to the minimal-sized solution, this means that finding k-size solution (or any solution) to a delete-free HTN problem ($\text{HTN}_{+\text{eff}}$) is in NP. Given that both plan-existence and k-length-plan-existence are NP-hard for $\text{HTN}_{+\text{eff}}$, the last of our results is trivial:

Theorem 5.11 *For $\text{HTN}_{+\text{eff}}$, both plan-existence and k-length-plan-existence are* NP*-complete.*

From this theorem and the subclass relationships shown in Figure 1, the other classes considered in this paper fall in NP as well and are thus NP-complete (with the exception of plan-existence for $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$, of course). Table 2 summarizes our final set of results.

Table 2: Summary of results after Section 5.

Problem	plan-existence	k-length-plan-existence
$\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$	P	NP-complete
$\text{TIHTN}_{+\text{eff}}$	NP-complete	NP-complete
$\text{HTN}^{+\text{pre}}_{+\text{eff}}$	NP-complete	NP-complete
$\text{HTN}_{+\text{eff}}$	NP-complete	NP-complete

6 Related Work

Erol, Hendler, and Nau's 1996 paper is the seminal work on HTN planning complexities, proving that while HTN planning is undecidable in general, many syntactic restrictions are decidable, with complexities ranging from polynomial time for problems with only totally ordered, primitive task networks to PSPACE-complete for so-called *regular* task networks, to in EXPTIME and 2-EXPTIME for problems with totally-ordered networks. (Alford et al. 2012) extends this work by showing larger subsets of HTN planning are decidable with search-based algorithms. Both these papers identify decidable subsets with restrictions on the structure of task networks in the domain, where as delete-free HTN planning allows arbitrary method structures, but restricts the form that operators can take.

In addition to the still-quite-high decision complexities, these task-network-identifiable subsets of HTN planning are unsuitable for use as heuristics, since in general, imposing a strict structure on the task networks of a problem removes solutions. Removing delete effects and negative preconditions increases the number of solutions to a problem, making a solution to this relaxed problem an underestimation of how hard it is to solve the original problem.

Geier and Bercher (2011) provides both the formalization of Task-Insertion HTN planning used in our paper and the decomposition trees and related lemmas about them used in our proofs. In their paper, they show that if any solution to a task-insertion problem exists, then there exists an acyclic decomposition tree which can be made executable through task insertion. Despite providing shorter decomposition trees than we used in our proofs (by a factor of $|O|$), there is no clear way to compress a decomposition tree when delete effects are allowed. This forces Geier and Bercher's witnesses to be full expanded, making them exponential in size. Although this provides an EXPSPACE upper bound on the complexity of task-insertion HTN planning, they do not provide a lower bound. Though it's obviously PSPACE-hard, a tighter bound on task-insertion HTN planning would be, if nothing else, intellectually satisfying.

Elkawkagy et al. (2012) provides a method for constructing *task landmark tables* which identify both the tasks that must occur in any primitive decomposition of a given task name, and the tasks that may occur. The landmark tables are insensitive to the state, making them similar to our state-transition preservability property that we introduced in the proof of Theorem 3.1 in the steps shown in the enumerated list there. Elkawkagy et al. use the landmark tables to provide upper and lower bounds on the computational cost of searching for a primitive executable decomposition. In principle, we could use state-transition preservability in a similar fashion by taking an HTN problem, removing all preconditions and effects from its operators, and running the state-transition preservability test. This would provide a lower bound on the size of any solution (making it an admissible heuristic) but would not bound the computational cost of finding such a solution. In practice, since state-transition preservability is decidable in NP, we would need to develop search control and heuristics techniques to be able to use it. However, our preliminary theoretical investigations suggest that under certain circumstances, this property can still be computed in polynomial time. For example, when there is only one state to be explored yet, such as the state generated by Relaxed Graphplan when it reaches to a fixpoint, we can generate the table in polynomial time via the second half of the procedure in the proof of Theorem 3.1. We will investigate this topic more in future work.

Finally, Alford, Kuter, and Nau (2009) described a technique for translating totally-ordered HTN problems into classical planning problems, but this technique requires giving an appropriate user-supplied bound during translation. Then, given a correct bound, the resulting classical problem is solvable iff the original HTN planning problem was solvable. In turn, still given the correct bound, any admissible classical heuristic can then be a ω-admissible HTN heuristic. One classical heuristic in specific, Relaxed Graphplan, may obviate the need for providing a correct translation bound, but it produces only polynomial sized plans. This may greatly underestimate the minimal size of a solution. One fix might be to run Relaxed Graphplan on the translated problem, and then run the state-transition preservability test on the fixpoint state that Relaxed Graphplan generates.

7 Conclusions

In classical planning, relaxing the planning problem by removing negative preconditions and effects has been quite useful in the development of efficiently computable search heuristics. Our results show that this relaxation will not—by itself—produce efficiently computable HTN planning heuristics, because the relaxed problems are NP-hard. Thus the development of search heuristics for HTN planning will require a new kind of problem relaxation.

The solution tables that we used in the proof of Lemma 5.5 are a data structure similar to planning graphs, and it might be possible to use them as a foundation to develop new heuristics and search techniques for generating compact witnesses. Such witnesses could be used to provide heuristic estimates of relaxed plan length. Furthermore, a solution table also exhibit similarities to that of a *chart* in *chart parsing* (Kay 1986; Earley 1986; Charniak, Goldwater, and Johnson 1998; Ji 1993), in both the way the solution table is generated and structured. This suggests that it might be possible to use the techniques to generate efficient parse-trees in chart parsing for witness inference.

Another approach may be to combine relaxed planning graphs with a relaxation of the constraints that HTN planning formalisms impose on the search process. For example, our results show that efficiently computable HTN planning problems can be produced by removing negative preconditions and effects, and also allowing task insertion (i.e., allowing the application of any executable operator, regardless of whether or not it is reachable by some decomposition). We suspect that this might relax the problem too much for the heuristic values to be useful. But we think it may be possible to develop more accurate yet efficiently computable heuristics by developing a principled compromise, e.g., by restricting the inserted tasks to those available in some decomposition of the current task. This would be an interesting topic for future research.

Acknowledgments. This work has been supported in part by ARO grant W911NF1210471, ONR grant N000141210430, ONR contract N00014-12-C-0239, and DARPA/AFRL contract FA8650-11-C- 7191. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred.

References

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. 2012. HTN problem spaces: Structure, algorithms, termination. In *Symposium on Combinatorial Search*.

Alford, R.; Kuter, U.; and Nau, D. 2009. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *International Joint Conference on Artificial Intelligence*, 1629–1634.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129:5–33.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69:165–204.

Charniak, E.; Goldwater, S.; and Johnson, M. 1998. *Edge-Based Best-First Chart Parsing*. Association for Computational Linguistics. 127–133.

Comon, H.; Dauchet, M.; Gilleron, R.; Löding, C.; Jacquemard, F.; Lugiez, D.; Tison, S.; and Tommasi, M. 2007. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata. Released October, 12th 2007.

Earley, J. 1986. An efficient context-free parsing algorithm. *Readings in natural language processing* 25 – 33.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving hierarchical planning performance by the use of landmarks. In *AAAI Conference on Artificial Intelligence*, 1763–1769.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *AAAI Conference on Artificial Intelligence*, volume 94, 1123–1128.

Erol, K.; Hendler, J.; and Nau, D. 1996. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence* 18:69–93.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *International Joint Conference on Artificial Intelligence*, 1955–1961.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2003. The Metric-FF planning system: Translating “ignoring delete lists” to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Ji, P. 1993. A tree approach for tolerance charting. *International Journal of Production Research* 31:1023–1033.

Kay, M. 1986. Algorithm schemata and data structures in syntactic processing. *Readings in natural language processing* 35–70.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *International Joint Conference on Artificial Intelligence*, 459–466.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Teichteil-Königsbuch, F.; Kuter, U.; and Infantes, G. 2010. Incremental plan aggregation for generating policies in MDPs. In *International Conference on Autonomous Agents and Multiagent Systems*, 1231–1238.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *International Conference on Automated Planning and Scheduling*, 352–359.