On the Decidability of HTN Planning with Task Insertion

Thomas Geier and **Pascal Bercher**
Ulm University, Institute of Artificial Intelligence, Ulm, Germany
forename.surname@uni-ulm.de

Abstract

The field of deterministic AI planning can roughly be divided into two approaches — classical state-based planning and hierarchical task network (HTN) planning. The plan existence problem of the former is known to be decidable while it has been proved undecidable for the latter. When extending HTN planning by allowing the unrestricted insertion of tasks and ordering constraints, one obtains a form of planning which is often referred to as "hybrid planning".

We present a simplified formalization of HTN planning with and without task insertion. We show that the plan existence problem is undecidable for the HTN setting without task insertion and that it becomes decidable when allowing task insertion. In the course of the proof, we obtain an upper complexity bound of $\mathbf{EXPSPACE}$ for the plan existence problem for propositional HTN planning with task insertion.

1 Introduction

Planning is known to be intractable in general. This holds both for classical state-based planning and for hierarchical planning. For the former, the complexity of the plan existence problem ("Is there a plan that solves the given planning problem?") reaches from constant time over $\mathbf{EXPSPACE}\mathbf{-complete}$ to undecidable [Erol *et al.*, 1995] depending on various restrictions. The most expressive "configuration" of classical planning that is still decidable[1] is thus intractable. Hierarchical task network (HTN) planning was created to mitigate this complexity problem by introducing a hierarchy over operators and thereby including search control knowledge into the planning process. Contrary to the intuition that the introduced hierarchy makes planning easier, it introduced undecidability even for the case in which classical planning is "only" $\mathbf{PSPACE}\mathbf{-complete}$[2]; this result was proved by Erol *et al.* [1996] for their formalization of HTN planning [Erol *et al.*, 1994].

The hierarchical aspect in HTN planning is achieved by introducing the concept of *compound tasks* in addition to the operators known from classical planning, which we call *primitive tasks* in this context. Compound tasks can be decomposed into predefined partial plans using so-called decomposition methods. The goal is to find an executable plan that was obtained by decomposing compound tasks in the initial plan until all tasks are primitive. Although the provision of decomposition methods can speed up the search process, they introduced undecidability. However, several restrictions can be imposed on methods which make the plan existence problem decidable: for instance if the initial task network and all methods are totally ordered or if all methods are acyclic [Erol *et al.*, 1996]. In this work, we show how decidability can be achieved without restricting methods by altering the solution criterion of HTN planning problems.

We investigate the decidability of the plan existence problem for *HTN planning with task insertion*, a planning formalism that lies between classical and HTN planning and is therefore often referred to as *hybrid planning*: the idea is to have an initial task network that needs to be decomposed, but to allow the insertion of tasks without requiring them to be inserted by the decomposition of compound tasks. This particular variant of fusing classical with HTN planning has been addressed before by Estlin *et al.* [1997], Schattenberg and Biundo [2006], and Gerevini *et al.* [2008]. The most important argument for this planning paradigm is that it overcomes a main criticism of HTN planning: *"HTN planning reduces the flexibility of an agent to respond to situations that are not anticipated by the writer of the task reduction schemas"* [Kambhampati *et al.*, 1998]. This argument is also the main motivation for Kambhampati *et al.*'s formalization of hybrid planning. There, compound tasks also show preconditions and effects like primitive ones and they can be inserted like operators in classical planning (which is also true for the work of Schattenberg and Biundo [2006]). However, Kambhampati *et al.* do not specify an initial task network. Hence, there is *no need* to insert and decompose compound tasks and therefore, they are only an efficiency boost for solving *classical* planning problems.

In the following sections we will first introduce our formalization of HTN planning with task insertion. We explain

[1]Datalog (no function symbols and finitely many constant symbols); operators are given in the input; allow delete lists and negated preconditions [Erol *et al.*, 1995].

[2]Propositional; operators are given in the input; allow delete lists and negated preconditions [Erol *et al.*, 1995].

why our formalism, although stripped of many technical aspects, is still susceptible to the original proof of undecidability for HTN planning when disallowing insertion of tasks. We then present a result that yields an upper bound on the length of shortest solutions and derive from it an upper complexity bound for the plan existence decision problem. In the end we conclude our paper with a discussion of the presented results.

2 Formalism

In this section, we introduce our notion of *HTN planning with task insertion* (or *hybrid planning*, for short). For the sake of simplicity, we base our formalism on a ground state representation. That is, there exist only boolean propositions. The choice effects only the final complexity class but not the decidability, as long as a potential first-order representation does not introduce undecidability on its own, e.g., by the use of functions.

During the rest of the paper, we consider relations and functions as sets of tuples. We use the bar notation for restricting relations and functions. Given a set of tuples $Q \subseteq R \times R$, we define $Q|_X := \{(q_1, q_2) \in Q \mid q_1, q_2 \in X\}$ and for a function $f : R \to S$, we define $f|_X := \{(r, s) \in f \mid r \in X\}$.

We begin by describing a task network, which represents a generalization of a task sequence in that it is only partially ordered.

Definition 1 (Task Network)**.** A *task network* $\mathrm{tn} = (T, \prec, \alpha)$ over a set of task names X is a tuple, where

- T is a finite and non-empty set of tasks
- $\prec \; \subseteq T \times T$ is a strict partial order on T (irreflexive, asymmetric, and transitive)
- $\alpha : T \to X$ labels every task with a task name

We write TN_X for the set of all task networks over the task names in X. Given a task network $\mathrm{tn} = (T, \prec, \alpha)$ and a set of tasks T', we define its restriction $\mathrm{tn}|_{T'} := (T \cap T', \prec|_{T'}, \alpha|_{T'})$. Also we write $T(\mathrm{tn})$ to refer to the tasks of tn.

Tasks serve the purpose of unique identifiers because task names (like "move" or "open door") may occur multiple times in the same task network. This leads us to the next definition: two task networks are isomorphic if they describe the same arrangement of task names despite using different identifiers.

We define two task networks $\mathrm{tn} = (T, \prec, \alpha)$ and $\mathrm{tn}' = (T', \prec', \alpha')$ as being *isomorphic*, written $\mathrm{tn} \cong \mathrm{tn}'$, if and only if there exists a bijection $\sigma : T \to T'$, such that for all $t, t' \in T$ it holds that $(t, t') \in \prec$ if and only if $(\sigma(t), \sigma(t')) \in \prec'$ and $\alpha(t) = \alpha'(\sigma(t))$.

Definition 2 (Planning Problem)**.** A *planning problem* is a 6-tuple $\mathcal{P} = (L, C, O, M, c_I, s_I)$ and

- L, a finite set of *proposition symbols*
- C, a finite set of *compound task names*
- O, a finite set of *primitive task names* with $C \cap O = \emptyset$
- $M \subseteq C \times TN_{C \cup O}$, a finite set of *decomposition methods*
- $c_I \in C$, the *initial task name*
- $s_I \in 2^L$, the *initial state*

For each $o \in O$, $(\mathrm{prec}(o), \mathrm{add}(o), \mathrm{del}(o)) \in 2^L \times 2^L \times 2^L$ is called the operator of o and consists of a precondition, an add-, and a delete list. By $\mathrm{tn}_I := (\{t_I\}, \emptyset, \{(t_I, c_I)\})$ we denote a fixed initial task network which consists only of the initial task t_I mapping to the initial task name c_I.

HTN planning formalizations usually feature an initial task network instead of only a single initial task name. We would like to note that our choice does not reduce the class of problems that can be expressed.

Both compound and primitive task names will be used to label the elements of task networks. We refer to tasks that are labeled with elements from C and O as *compound tasks* and *primitive tasks*, respectively. We refer to a task network over O as a *primitive task network*.

As during the rest of this paper there is always exactly one planning problem under consideration, we do not write out "$\mathcal{P} = (L, C, O, M, c_I, s_I)$" each time. Thus, whenever any of the symbols L, C, O, M, c_I, and s_I are used, then they are part of the planning problem $\mathcal{P}$.

After we have defined the basic components of a planning problem, we continue by describing how an initial task network can be transformed into a solution by decomposition and task insertion.

The decomposition of a compound task results in its removal from the task network, followed by an insertion of a copy of the method's task network. The ordering constraints on the removed task are inherited by its replacement tasks (cf. the set $\prec_X$ in the next definition).

Definition 3 (Decomposition)**.** A method $m = (c, \mathrm{tn}_m)$ decomposes a task network $\mathrm{tn}_1 = (T_1, \prec_1, \alpha_1)$ into a new task network tn_2 by replacing task t, written $\mathrm{tn}_1 \xrightarrow[t,m]{} \mathrm{tn}_2$, if and only if $t \in T_1$, $\alpha_1(t) = c$, and there exists a task network $\mathrm{tn}' = (T', \prec', \alpha')$ with $\mathrm{tn}' \cong \mathrm{tn}_m$ and $T' \cap T = \emptyset$, and[3]

$$\begin{aligned} \mathrm{tn}_2 &:= ((T_1 \setminus \{t\}) \cup T', \prec_1 \cup \prec' \cup \prec_X, \alpha_1 \cup \alpha') \text{ with} \\ \prec_X &:= \{(t_1, t_2) \in T_1 \times T' \mid (t_1, t) \in \prec_1\} \cup \\ &\quad\;\; \{(t_1, t_2) \in T' \times T_1 \mid (t, t_2) \in \prec_1\} \end{aligned}$$

Given a planning problem $\mathcal{P}$, we write $\mathrm{tn}_1 \to_D^* \mathrm{tn}_2$, if tn_2 can be decomposed from tn_1 by an arbitrary number of decompositions using methods from M.

Note that $\to_D^*$ is reflexive and transitive. Also note that the constructed set of ordering constraints $\prec_1 \cup \prec' \cup \prec_X$ is again a strict partial order.

Other formalizations of hybrid planning complement decomposition with the insertion of both primitive and compound tasks, where compound tasks also show preconditions and effects [Kambhampati *et al.*, 1998; Schattenberg and Biundo, 2006]. Since the result of inserting a compound task can be simulated by inserting a decomposed version directly, we only allow the insertion of primitive tasks.

Definition 4 (Task Insertion)**.** Given a task network $\mathrm{tn}_1 = (T_1, \prec_1, \alpha_1)$ and a primitive task name o, then a task network tn_2 can be obtained from tn_1 by insertion of o, if and only if $\mathrm{tn}_2 = (T_1 \cup \{t\}, \prec_1, \alpha_1 \cup \{(t, o)\})$ for some $t \notin T_1$.

[3]For correctness, ordering and labeling of tn_2 must be restricted to $(T_1 \setminus \{t\}) \cup T'$; omitted for better readability.

A task network $\mathrm{tn}_2 = (T_1, \prec_2, \alpha_1)$ can be obtained from tn_1 by insertion of an ordering constraint (t, t'), if and only if $t, t' \in T_1$ and $\prec_2$ is minimal and a strict partial order such that $\prec_1 \cup \{(t, t')\} \subseteq \prec_2$.

Given a planning problem $\mathcal{P}$, we say that tn_2 can be obtained from tn_1 via *task insertion*, written $\mathrm{tn}_1 \rightarrow_I^* \mathrm{tn}_2$, if tn_2 can be obtained from tn_1 by an arbitrary number of insertions of orderings and primitive task names from O.

Note that $\rightarrow_I^*$ is reflexive and transitive.

Definition 5 (Executable Task Network)**.** Given a planning problem $\mathcal{P}$, then a task network $\mathrm{tn} = (T, \prec, \alpha)$ is *executable in state* $s \in 2^L$, if and only if it is primitive and there exists a linearization of its tasks $t_1, \ldots, t_n$ that is compatible with $\prec$ and a sequence of states $s_0, \ldots s_n$ such that $s_0 = s$ and $\mathrm{prec}(\alpha(t_i)) \subseteq s_{i-1}$ and $s_i = (s_{i-1} \setminus \mathrm{del}(\alpha(t_i))) \cup \mathrm{add}(\alpha(t_i))$ for all $1 \leq i \leq n$. We call s_n *the state generated by* tn.

Definition 6 (Solution)**.** A task network tn_S is a solution to a planning problem $\mathcal{P}$, if and only if (1) tn_S is executable in s_I and (2) $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_S$ for tn_S being an *HTN* solution to $\mathcal{P}$ or (2') there exists a task network tn_B such that $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_B \rightarrow_I^* \mathrm{tn}_S$ for tn_S being a *hybrid* solution to $\mathcal{P}$. $\mathrm{Sol}_{\text{HTN}}(\mathcal{P})$ and $\mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$ denote the set of all HTN and hybrid solutions of $\mathcal{P}$, respectively.

For the purpose of this paper, we refer to *hybrid solutions* as *solutions* and use the term *HTN solutions* if necessary.

Note that the presented hybrid formalism is capable of capturing ground classical planning. The only missing piece in comparison to classical planning is the goal condition. This can be simulated by having an initial task network that forces an artificial final task, carrying the goal condition in its precondition as the last task in every solution.

3 On the (Un-)Decidability of HTN Planning

When using the HTN solution criterion, we talk about HTN planning, rather than hybrid planning. Although our formalization of HTN planning is inspired by Erol *et al.*'s formalization [1994], it is still a simplification of the latter. As we are going to show the decidability of our hybrid planning formalism, the question arises whether this is due to these simplifications or due to the addition of task insertion. To address this question we reproduce Erol *et al.*'s proof of the undecidability of the plan existence problem for HTN planning [Erol *et al.*, 1996] using *our* formalization, which requires only minor adaptations. In particular, their proof relies on the usage of truth constraints, which are not available in our formalization. And it turns out that they are not necessary for the proof.

We begin by defining the plan existence problem for HTN planning as the set of planning problems that possess an HTN solution, i.e., $\text{PLAN-EXISTENCE}_{\text{HTN}} := \{\mathcal{P} \mid \mathcal{P}$ is a planning problem and $\mathrm{Sol}_{\text{HTN}}(\mathcal{P}) \neq \emptyset\}$.

Theorem 1. *PLAN-EXISTENCE*$_{\text{HTN}}$ *is undecidable*

The proof bases on the fact that one can imitate the production rules of context-free grammars (CFGs) by using decomposition methods. We reduce the undecidable question of whether the languages of two CFGs have a non-empty intersection [Hopcroft *et al.*, 2000, page 407] to the plan existence problem. The idea is to start with a task network that contains the start symbols from both grammars in parallel. These are decomposed into words from the respective grammar's language. The pre- and postconditions of the primitive tasks enforce a final plan in which the symbols of both words are mixed together like the teeth of a zipper. This allows us to ensure that the decomposed task network is executable if and only if the produced words are equal.

Proof. Let G_1, G_2 be two CFGs in Chomsky normal form (each rule has either the form $X \rightarrow YZ$ or $X \rightarrow a$, where X, Y, and Z are non-terminals and a is a terminal) defined over the same set of terminal symbols Σ. For $i \in \{1, 2\}$, we denote by Γ_i the non-terminal symbols of G_i with $S_i \in \Gamma_i$ being the start symbol of G_i. We assume $\Gamma_1 \cap \Gamma_2 = \emptyset$ and define $\Gamma := \Gamma_1 \cup \Gamma_2$. By $L(G_i)$ we denote the language generated by G_i and assume $\varepsilon \notin L(G_1) \cup L(G_2)$. We refer to the grammar rules of G_i by R_i and set $\mathrm{R} := \mathrm{R}_1 \cup \mathrm{R}_2$. We construct $\mathcal{P} = (L, C, O, M, c_I, s_I)$ with $c_I \notin \Gamma$ as follows:

- $L := \{turn_{G_1}, turn_{G_2}\} \cup \Sigma$
- $C := \Gamma \cup \{c_I\}$, $O := \{G_i^a \mid a \in \Sigma,\ i \in \{1, 2\}\} \cup \{F\}$
- $M := \{(c_I, \mathrm{tn})\} \cup$
  $\{(X, \mathrm{tn}_{X \rightarrow YZ}) \mid X \rightarrow YZ \in \mathrm{R}\} \cup$
  $\{(X, \mathrm{tn}_{X \rightarrow a}) \mid X \rightarrow a \in \mathrm{R}_i, i \in \{1, 2\}\}$, where
  $$\begin{aligned} \mathrm{tn} &:= (\{s_1, s_2, f\}, \{(s_1, f), (s_2, f)\}, \\ &\qquad \{(s_1, S_1), (s_2, S_2), (f, F)\}) \\ \mathrm{tn}_{X \rightarrow YZ} &:= (\{t, t'\}, \{(t, t')\}, \{(t, Y), (t', Z)\}) \\ \mathrm{tn}_{X \rightarrow a} &:= (\{t\}, \emptyset, \{(t, G_i^a)\}) \end{aligned}$$
- $s_I := \{turn_{G_1}\}$

The operator of $G_1^a \in O$ is defined as $(\{turn_{G_1}\}, \{turn_{G_2}, a\}, \{turn_{G_1}\})$ and that of $G_2^a \in O$ as $(\{turn_{G_2}, a\}, \{turn_{G_1}\}, \{turn_{G_2}, a\})$, respectively. This construction ensures strict turn-taking and a matching of the produced words. The operator of F is defined as $(\{turn_{G_1}\}, \emptyset, \emptyset)$; it ensures that the plan finishes with an operator of the second grammar. By construction, there is a word $\omega \in L(G_1) \cap L(G_2)$ if and only if $\mathcal{P}$ has a solution. $\square$

We have shown that one can express the CFG intersection problem within our HTN formalization, thus proving the undecidability of our planning formalism without insertion. Please note that a solution to the same planning problem using the *hybrid* solution criterion can produce solutions which do *not* correspond to words in the intersection of $L(G_1)$ and $L(G_2)$ due to the the arbitrary insertion of tasks.

4 On the Decidability of HTN Planning with Task Insertion

We now present our proof that yields an upper bound on the length of shortest solutions to a given planning problem. The following paragraph summarizes its main idea.

Suppose there exists a solution tn_S. This solution features a decomposition tn_B of the initial task network. In analogy to the pumping lemma for CFGs [Hopcroft *et al.*, 2000, page 274 ff.], we show that there exists also a "short" decomposition tn'_B. It is obtained by "pumping-down" tn_B, removing all parts that were produced by cycles in the decomposition process. We show that tn'_B can be developed into tn_B by task insertion, which in turn can be developed into tn_S.

Further we provide an upper bound on the number of tasks that must be inserted in order to turn a given task network into a solution, if possible. Applying this result to the short decomposition tn'_B, we can prove the existence of a short solution tn'_S. So in order to decide whether a solution to a planning problem exists, it suffices to check all task networks of a certain size.

4.1 Representing Task Decomposition

This section transfers the idea of a parse tree from formal grammars to the area of task network decompositions, where we call it a *decomposition tree*. It is a representation of how the initial compound task can be decomposed into a primitive task network.

Definition 7 (Decomposition Tree)**.** Given a planning problem $\mathcal{P}$, then a *decomposition tree* $g = (T, E, \prec, \alpha, \beta)$ is a five-tuple with the following properties. (T, E) is a tree with nodes T and directed edges E pointing towards the leafs. There is a strict partial order defined over the nodes, given by $\prec$. The nodes are labeled with task names by $\alpha : T \to C \cup O$. Additionally $\beta : T \to M$ labels inner nodes with methods.

We write $T(g)$ to refer to the tasks of g and $ch(g,t)$ to refer to the direct children of $t \in T(g)$ in g.

The following definition states under which circumstances a decomposition tree encodes a decomposition of the initial compound task. Note that a task network resulting from such a decomposition is *not* necessarily executable.

Definition 8 (Validity of Decomposition Tree)**.** A decomposition tree $g = (T, E, \prec, \alpha, \beta)$ is *valid* with respect to a planning problem $\mathcal{P}$, 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 $(c, \mathrm{tn}_m) := \beta(t)$, the following holds:

1. $\alpha(t) = c$
2. the task network induced in g by $ch(g,t)$ and tn_m are isomorphic, i.e., $(ch(g,t), \prec|_{ch(g,t)}, \alpha|_{ch(g,t)}) \cong \mathrm{tn}_m$
3. for all $t' \in T$ and all $c' \in ch(g,t)$ it holds that
   (a) if $(t, t') \in \prec$ then $(c', t') \in \prec$
   (b) if $(t', t) \in \prec$ then $(t', c') \in \prec$
4. there are no ordering constraints in $\prec$ other than those demanded by either 2. or 3.

The first criterion ensures the applicability of the methods, the inner nodes are labeled with; the second criterion ensures that the method's task networks are correctly represented within the decomposition tree; and the third criterion ensures the inheritance of ordering constraints as demanded by Definition 3. Please note that every task is uniquely used by g, as we require (T, E) to be a tree.

Definition 9 (Yield of Decomposition Tree)**.** The yield of a decomposition tree $g = (T, E, \prec, \alpha, \beta)$, written $\mathrm{yield}(g)$, is a task network defined as follows. Let $T' \subseteq T$ be the set of all leaf nodes of g. Then, $\mathrm{yield}(g) := (T', \alpha|_{T'}, \prec|_{T'})$.

Proposition 1. *Given a planning problem $\mathcal{P}$, then for any task network $\mathrm{tn} \in TN_{C \cup O}$ the following holds. There exists a valid decomposition tree g with $\mathrm{yield}(g) = \mathrm{tn}$, if and only if $\mathrm{tn}_I \to_D^* \mathrm{tn}$.*

Proof Sketch. For the forward implication, make an induction over the number of inner nodes of g. Note that each inner node corresponds to one method application. As base case, we have the valid decomposition tree without inner nodes, that must consist of only one task that is labeled with c_I and hence its yield is the initial task network tn_I. For the inductive step, fix a valid decomposition tree g with $n+1$ inner nodes. Let t be an inner node, for which all children $ch(g,t)$ are leafs. Then consider the tree g' which has all children of t removed. Show that g' is valid based on the validity of g. Then show that $\mathrm{yield}(g') \xrightarrow[t,\beta(t)]{} \mathrm{yield}(g)$.

For the backward implication, make an induction over the length of the decomposition sequence. The base case holds due to the reflexivity of $\to_D^*$. In the inductive step, construct the tree by adding newly inserted tasks as children of the replaced task. $\square$

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

The result of this operation as defined by us is only reasonable for our particular use-case. A general subtree substitution operation would have to create an isomorphic copy of the inserted subtrees, which we have omitted for simplicity.

Definition 10 (Subtree Substitution)**.** Let $g = (T, E, \prec, \alpha, \beta)$ be a decomposition tree and $t_i, t_j \in T$ be two nodes of g. If t_i is the root node of g, then we define the result of the subtree substitution on g that substitutes t_i by t_j, written $g[t_i \leftarrow t_j]$, as $g[t_i \leftarrow t_j] := g[t_j]$; 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

The following proposition states that the result of a subtree substitution still describes decompositions if applied under the right circumstances. Also refer to Figure 1 for an illustration of the operation.

Proposition 2. *Given a valid decomposition tree $g = (T, E, \prec, \alpha, \beta)$ with respect to a planning problem $\mathcal{P}$ and two nodes $t_i \in T$, $t_j \in T(g[t_i])$ with $\alpha(t_i) = \alpha(t_j)$, then $g[t_i \leftarrow t_j]$ is also a valid decomposition tree with respect to $\mathcal{P}$.*

Proof Sketch. We have to show that $g' := g[t_i \leftarrow t_j] = (T', E', \prec', \alpha', \beta')$ is still a tree, its root node is labeled with c_I, and that $\prec$ is minimal such that the first three criteria of Definition 8 hold for all inner nodes of g'. For the remainder, let $p \in T$ be such that $(p, t_i) \in E$.

For the special case of the subtree substitution, in which t_i is the root node of g, there is nothing to show, since $g' = g[t_j]$ is clearly a valid decomposition tree with respect to $\mathcal{P}$. Thus, let t_i be different from the root node. It is also easy to see that g' is a tree and its root node is labeled with c_I.

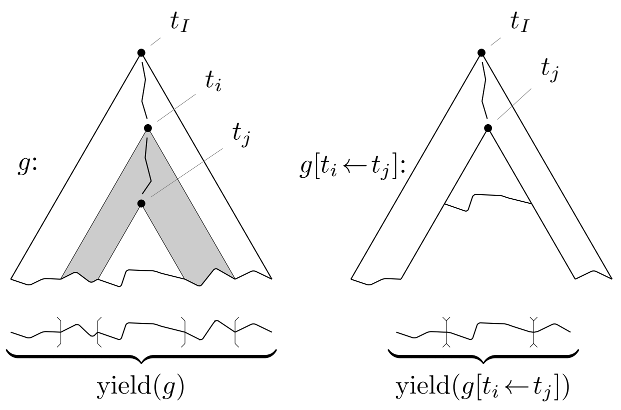

Figure 1: On the left, a decomposition tree g is depicted. The result of the subtree substitution that replaces the subtree below t_i with the subtree below t_j is shown on the right. The gray area inside g corresponds to the tasks that get removed during the substitution. Note how the subtree substitution affects the yield of the tree by removing the contribution of the gray area.

Criterion 1 holds for all inner nodes of g' as both α' and β' are defined on the same nodes and $\alpha' \subseteq \alpha$ and $\beta' \subseteq \beta$.

For criterion 2, we fix some inner node $t \in T'$ and consider the two cases that either $t \neq p$ or $t = p$. The case in which $t \neq p$ is straight forward as the task network induced by $ch(g, t)$ in g is the same as the one induced by $ch(g', t)$ in g'. If $t = p$, the induced task network in g' is still isomorphic to the one in g, because we substituted t_i by t_j and $\alpha(t_i) = \alpha(t_j)$.

To show criterion 3 (considering only 3a, as 3b is analogous), take some ordering constraint (t, t') in $\prec'$. It must also occur in $\prec$ and thus $(c', t') \in \prec$ for all children c' of t. Show $(c', t') \in \prec'$ if $c' \neq t_i$ and $(t_j, t') \in \prec'$, otherwise.

Proving the minimality of $\prec'$ is the most difficult part. We have to show for each $(t_1, t_2) \in \prec'$ that it is required by criterion 2 or 3. Consider the cases that $(t_1, t_2) \in \prec$ is originally produced by criterion 2 (case I) or 3 (case II).

For case I, show that as a consequence (t_1, t_2) is required in $\prec'$ because of criterion 2 with special treatment for $t_1, t_2 \in ch(g', p)$. Intuitively all decompositions in g' have counterparts in g and they produce the same ordering in both trees.

For proving case II (again, we consider only 3a), let $(c', t') := (t_1, t_2)$. We know that there exists a node $t \in T$ with $(t, c') \in E$ and $(t, t') \in \prec$, as this is the antecedent of criterion 3a. Now consider the cases of $t \in T'$ (case IIa) and $t \notin T'$ (case IIb). The proof for case IIa is straight forward. For case IIb, we can conclude that $c' = t_j$ since t_j is the only node in g' that has lost its parent in g. Show that $(p, t') \in \prec'$ and use this as premise to apply criterion 3a in order to show that (c', t') must be in $\prec'$.

□

4.2 Bounding Solution Sizes by Eliminating Cycles

In order to prove our main theorem, which establishes a size limit on the smallest solution of a planning problem, we first need to prove three lemmas. The first one allows us to shorten a decomposition sequence that contains a cycle over compound task names. When doing so, we remain able to recreate via task insertion what has been removed. This result is used by the second lemma to give an upper bound on the size of shortest task networks originated from decomposition that can possibly be developed to a solution. Then the third lemma allows us to find a limit to the number of tasks that have to be inserted in order to turn a given task network into a solution.

The proof of the first lemma uses the idea of the proof of the pumping lemma for CFGs to claim the existence of a shorter version of a given decomposition.

Lemma 1. *Given a planning problem $\mathcal{P}$ and a primitive task network* tn *with valid decomposition tree g, for which there exists a path $t_1, \ldots, t_n$ in g with $\alpha(t_i) = \alpha(t_j)$ for some i and j with $i < j \leq n$ (the path contains a cycle). Then it holds that* $\mathrm{tn}_I \rightarrow_D^* \mathrm{yield}(g[t_i \leftarrow t_j]) \rightarrow_I^* \mathrm{tn}$.

Proof. Fix the premises from the lemma. We notice that the conditions for applying Proposition 2 are given. Thus, we can state that $g[t_i \leftarrow t_j]$ is valid. Therefore it generates a task network, name it $\mathrm{tn}' = (T', \prec', \alpha')$, that can be obtained from tn_I by decomposition. It remains to show that $\mathrm{tn}' \rightarrow_I^* \mathrm{tn}$. Since tn' consists of the leafs of $g[t_i \leftarrow t_j]$ and $\mathrm{tn} = (T, \prec, \alpha)$ is induced by the leafs of g, and the substitution only takes away nodes and ordering constraints from g and does not create new leaf nodes, clearly $T' \subseteq T$ and $\prec' \subseteq \prec$ and also $\alpha|_{T'} = \alpha'$. We can thus obtain tn from tn' by adding additional tasks and ordering constraints. □

By using this result repeatedly, we are now able to remove all cycles from a decomposition tree. This makes it possible to formulate an upper bound on the size of the tree and thus of the generated task network.

Lemma 2. *Given a planning problem $\mathcal{P}$, then for every task network* tn *with* $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}$, *there exists a task network* tn' *with* $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}' \rightarrow_I^* \mathrm{tn}$ *and* $|T(\mathrm{tn}')| \leq b^{|C|}$, *where b is the number of tasks inside the largest task network of the methods from M.*

Proof. Fix a planning problem $\mathcal{P}$ and a task network tn with $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}$. Fix a decomposition tree g of tn. If g contains a cycle on one of its paths (i.e., two nodes labeled with the same compound task symbol), then Lemma 1 allows us to remove it, obtaining a new decomposition tree g_1 and corresponding task network $\mathrm{tn}_1 = \mathrm{yield}(g_1)$ with $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_1 \rightarrow_I^* \mathrm{tn}$.

As long as the new decomposition tree still contains a cycle, we can repeat the procedure. By doing so, we obtain a sequence of decomposition trees $g_1, \ldots, g_n$ and corresponding task networks $\mathrm{tn}_1, \ldots, \mathrm{tn}_n$. The last decomposition tree g_n does not contain a cycle on any of its paths. The sequence is finite, since every subtree substitution removes at least one task, thus $n \leq |T(g)|$. For the sequence of task networks, it holds that $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_n \rightarrow_I^* \mathrm{tn}_{n-1} \rightarrow_I^* \ldots \rightarrow_I^* \mathrm{tn}_1 \rightarrow_I^* \mathrm{tn}$ and by the transitivity of task insertion we get $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_n \rightarrow_I^* \mathrm{tn}$. We choose tn_n as the tn' from the lemma. It remains to show the size bound $|T(\mathrm{tn}_n)| \leq b^{|C|}$.

The size of tn_n is limited because its decomposition tree g_n contains no cycle on any of its paths and is thus bounded in depth by $|C|$. Taking the maximal branching factor b of g, we get an upper bound of $b^{|C|}$ of the number of leaf nodes of tn_n and thus on the number of tasks of $\mathrm{tn}_n = \mathrm{yield}(g)$. $\square$

While the first two lemmas have somehow tamed the decomposition aspect of the solution criteria, we still need to care for executability. The idea is to take the tasks that have been introduced by decomposition and connect them in an executable way by using task insertion. Thus we obtain one classical planning problem for each task inside the decomposed task network.

Proposition 3. *Let $\mathcal{P}$ be a planning problem and $s, s' \in 2^L$. If there exists a task network* tn *which is executable in s and generates s', then there is also a task network* tn' *which is executable in s, generates s', and $|T(\mathrm{tn}')| \leq 2^{|L|}$.*

Proof. As the size of the state space induced by $\mathcal{P}$ is $2^{|L|}$, each task network longer than this value must traverse at least one state twice and hence must contain a cycle in the task sequence which can simply be omitted. $\square$

Lemma 3. *Given a planning problem $\mathcal{P}$ and a task network* tn*, that can be developed into a solution $\mathrm{tn}_S \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$ via task insertion $\mathrm{tn} \rightarrow_I^* \mathrm{tn}_S$, then there exists a solution $\mathrm{tn}'_S \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$ with $|T(\mathrm{tn}'_S)| \leq |T(\mathrm{tn})|\,(2^{|L|} + 1)$.*

Proof. Since tn_S is a solution to $\mathcal{P}$, there exists an executable linearization Lin_S of its tasks. Let $\mathrm{Lin} := t_1, t_2, \ldots, t_n$ be the tasks of tn ordered as they appear inside Lin_S. Let s_i and s'_i be the states before and after the execution of t_i when applying Lin_S to the initial state s_I.

In order to develop Lin into an executable task sequence, we need to find the $n-1$ task networks tn_i that transform s'_i into s_{i+1} for $0 < i < n-1$ and the task network tn_0 that transforms s_I into s_1. Proposition 3 tells us that we can find such task networks containing at most $2^{|L|}$ many tasks each, if these problems are solvable at all. And we can use Lin_S as a witness, that they are solvable.

Putting it all together, we can construct a solution task network illustrated by the sequence $\mathrm{tn}_0 t_1 \mathrm{tn}_1 t_2 \ldots \mathrm{tn}_{n-1} t_n$. This task network contains at most $n2^{|L|} + n$ many tasks and can be constructed from tn via task insertion. $\square$

Theorem 2. *Given a planning problem $\mathcal{P}$ with $\mathrm{Sol}_{\text{HYBRID}}(\mathcal{P}) \neq \emptyset$, then there exists a solution $\mathrm{tn}^*_S \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$ with $|T(\mathrm{tn}^*_S)| \leq b^{|C|}(2^{|L|} + 1)$, where b is the number of tasks inside the largest task network of the methods from M.*

Proof. A graphical presentation of this proof is given in Figure 2. Fix a solution $\mathrm{tn}_S \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$. There exists a task network tn_B with $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_B$ and $\mathrm{tn}_B \rightarrow_I^* \mathrm{tn}_S$ because of the solution criteria. This task network marks the boundary between the decomposition part and the insertion part of the planning process. By Lemma 2 there exists a (not necessarily different) task network tn'_B with $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}'_B \rightarrow_I^* \mathrm{tn}_B$ and $|T(\mathrm{tn}'_B)| \leq b^{|C|}$. Because of transitivity of task insertion, from $\mathrm{tn}'_B \rightarrow_I^* \mathrm{tn}_B \rightarrow_I^* \mathrm{tn}_S$, it follows that $\mathrm{tn}'_B \rightarrow_I^* \mathrm{tn}_S$.

Using Lemma 3, from $\mathrm{tn}'_B \rightarrow_I^* \mathrm{tn}_S$, it follows that there exists a solution tn^*_S with $|T(\mathrm{tn}^*_S)| \leq |T(\mathrm{tn}'_B)|\,(2^{|L|} + 1)$ and thus $|T(\mathrm{tn}^*_S)| \leq b^{|C|}(2^{|L|} + 1)$. $\square$

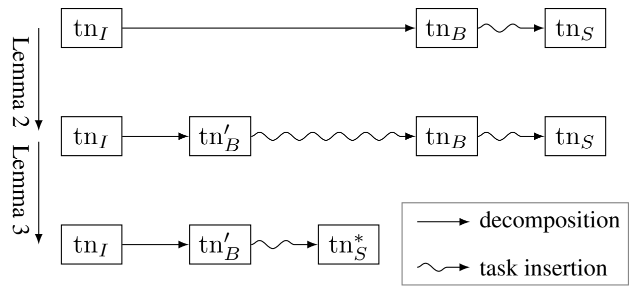

Figure 2: This figure shows how the proof of Theorem 2 constructs a small solution tn^*_S from a large solution tn_S. The task network tn_B marks the boundary between task decomposition and task insertion. The length of the arrows corresponds to the number of applied modifications, and thus correlates to the size of the resulting task network.

4.3 Complexity Results

In this section we will put the previous results to use in order to obtain an upper complexity bound of $\mathbf{EXPSPACE}$ for the plan existence problem for propositional, hybrid planning.

First we are going to define two decision problems. As usual, these are represented by subsets of words over a given alphabet Σ. We do not deal with the problem of encoding the syntactical structures, such as task networks or problems, into words over Σ. It is assumed that this process and the rejection of malformed inputs is feasible and does not add to the complexity of the problem. We denote the length of the description of an object X in Σ by $|X|_\Sigma$. As a remark about task networks, note that the description of a task network tn is bounded by $O(|T(\mathrm{tn})|^2)$ because of the possible number of ordering constraints.

We define the decision problem SOLUTION$_{\text{HYBRID}}$ as all tuples of a planning problem and a corresponding hybrid solution, i.e., SOLUTION$_{\text{HYBRID}} := \{(\mathcal{P}, \mathrm{tn}) \mid \mathcal{P}$ is a planning problem and $\mathrm{tn} \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})\}$. Then we can state the following proposition.

Proposition 4. *SOLUTION*$_{\text{HYBRID}} \in \mathbf{PSPACE}$.

Proof. Take as input the tuple $(\mathcal{P}, \mathrm{tn})$. Since we demand that a task network may not be empty, the application of a decomposition to a task network never decreases its number of tasks $|T(\mathrm{tn})|$. This means that during a decomposition $\mathrm{tn}_1 \rightarrow_D^* \mathrm{tn}_2$, the intermediate task networks never exceed $\mathrm{pol}(|\mathrm{tn}_2|_\Sigma)$. Obviously the same holds for a sequence of insertions. We can thus state the following non-deterministic algorithm, that runs in polynomial space in the size of $|(\mathcal{P}, \mathrm{tn})|_\Sigma$ and thus in $\mathbf{PSPACE}$. (1) Check if tn is executable in s_I; reject if not. (2) The following works in-place: non-deterministically decompose tn_I into a primitive task network tn' with size of at most $\mathrm{pol}(|\mathrm{tn}|_\Sigma)$; then, non-deterministically insert tasks and ordering constraints not violating this size bound; check if tn was produced; accept if so, otherwise reject. $\square$

We define the plan existence problem for hybrid planning as the set of planning problems that possess a hybrid solution, i.e., PLAN-EXISTENCE$_{\text{HYBRID}}$:= $\{\mathcal{P} \mid \mathcal{P}$ is a planning problem and $\text{Sol}_{\text{HYBRID}}(\mathcal{P}) \neq \emptyset\}$.

Corollary 1. *PLAN-EXISTENCE*$_{\text{HYBRID}} \in \mathbf{EXPSPACE}$.

Proof. Let the input be $\mathcal{P}$ and let $n = |\mathcal{P}|_\Sigma$ be the length of this input. Theorem 2 states that there exists a solution tn_S^* with $|T(\text{tn}_S^*)| \leq b^{|C|}(2^{|L|}+1)$, if and only if $\mathcal{P}$ has a solution at all. From the bound on the number of tasks and by substituting b, $|C|$, and $|L|$ by n, we can estimate $|\text{tn}_S^*|_\Sigma \leq \text{pol}(n^n(2^n+1))$. Thus, $|\text{tn}_S^*|_\Sigma \leq 2^{\text{pol}(n)}$ and the following non-deterministic algorithm runs in exponential space. (1) Non-deterministically guess a task network tn with $|T(\text{tn})| \leq b^{|C|}(2^{|L|}+1)$. (2) Check whether $(\mathcal{P}, \text{tn}) \in$ SOLUTION$_{\text{HYBRID}}$ using only space polynomial in $|(\mathcal{P}, \text{tn})|_\Sigma \leq n + 2^{\text{pol}(n)}$; accept if the check returns true, otherwise reject. The given algorithm decides PLAN-EXISTENCE$_{\text{HYBRID}}$ using only space exponential in the input length. □

And as a direct consequence, we conclude:

Corollary 2. *PLAN-EXISTENCE*$_{\text{HYBRID}}$ *is decidable.* □

5 Conclusion and Discussion

We have formalized a simplified, propositional version of HTN planning with task insertion (or hybrid planning) that allows for the insertion of tasks and ordering constraints. We have established $\mathbf{EXPSPACE}$ as an upper complexity bound of the corresponding plan existence problem. We have also shown that plan existence is undecidable given our formalization *without insertion*. We conclude that the possibility to insert tasks as an addition to task decomposition greatly reduces the computational complexity of HTN planning.

A direct application of the result to the established HTN formalism by Erol *et al.* [1994] might not be possible, since undecidability could have been "reintroduced" by a feature that is not captured by us, like the formulation of truth constraints. But even then we would have eliminated at least one source of undecidability.

Schattenberg and Biundo [2006] solve hybrid planning problems close to the ones defined in this paper. However, the initial task network and the decomposition methods may contain causal links, thereby further restricting the set of possible solutions. Thus, it is not clear, whether our decidability result also applies to their formalization.

However, our result seems to apply to the problems solved by the `Duet` planning system [Gerevini *et al.*, 2008]. It processes problems as defined in this paper, but uses a lifted state representation and features preconditions for decomposition methods. `Duet` could be extended to prune plans that exceed a certain size and achieve guaranteed termination without sacrificing completeness. Note that the complexity result is not directly transferable, since the step from a propositional state representation to a lifted one will most likely result in an exponential increase in complexity.

Since the focus of the paper lies on showing decidability, we did not attempt to provide tight complexity bounds for the plan existence problem. We have shown $\mathbf{EXPSPACE}$ membership and can state $\mathbf{PSPACE\text{-}hard}$ as a trivial lower bound. This is the case, because our formalization captures ground, classical planning with negative effects and operators given in the input, which has been proved to be $\mathbf{PSPACE\text{-}complete}$.

Acknowledgements

We want to thank our reviewers for their insightful questions and suggestions that helped us improving this paper.

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994)*, pages 249–254, 1994.

[Erol *et al.*, 1995] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76:75–88, 1995.

[Erol *et al.*, 1996] Kutluhan Erol, James Hendler, and Dana S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[Estlin *et al.*, 1997] Tara A. Estlin, Steve A. Chien, and Xuemei Wang. An argument for a hybrid HTN/operator-based approach to planning. In *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, pages 182–194, 1997.

[Gerevini *et al.*, 2008] Alfonso Gerevini, Ugur Kuter, Dana S. Nau, Alessandro Saetti, and Nathaniel Waisbrot. Combining domain-independent planning and HTN planning: The duet planner. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 573–577. IOS Press, 2008.

[Hopcroft *et al.*, 2000] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, volume 3. Addison-Wesley Reading, MA, second edition, 2000.

[Kambhampati *et al.*, 1998] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*, pages 882–888. AAAI Press, 1998.

[Schattenberg and Biundo, 2006] Bernd Schattenberg and Susanne Biundo. A unifying framework for hybrid planning and scheduling. In *Advances in Artificial Intelligence, Proceedings of the 29th German Conference on Artificial Intelligence (KI 2006)*, pages 361–373. Springer, 2006.