Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling

On the Complexity of HTN Plan Verification and Its Implications for Plan Recognition

Gregor Behnke, Daniel Höller, and **Susanne Biundo**
Ulm University, Institute of Artificial Intelligence, Ulm, Germany
{gregor.behnke, daniel.hoeller, susanne.biundo}@uni-ulm.de

Abstract

In classical planning it is easy to verify if a given sequence of actions is a solution to a planning problem. It has to be checked whether the actions are applicable in the given order and if a goal state is reached after executing them. In this paper we show that verifying whether a plan is a solution to an HTN planning problem is much harder. More specifically, we prove that this problem is **NP-complete**, even for very simple HTN planning problems. Furthermore, this problem remains **NP-complete** if an executable sequence of tasks is already provided. HTN-like hierarchical structures are commonly used to represent plan libraries in plan and goal recognition. By applying our result to plan and goal recognition we provide insight into its complexity.

1 Introduction

In classical planning it is polynomial to test whether a plan is a solution to a planning problem. This problem is called *plan verification*. It involves checking executability and if a goal is reached. Since hierarchical planning has an additional solution criterion, a more elaborated test is necessary. The criterion requires a solution to be obtained via decomposing the initial task network. It makes the plan existence problem in hierarchical planning **semi-decidable** in general, though there are subclasses that remain decidable (Erol, Hendler, and Nau 1996; Alford, Bercher, and Aha 2015). This raises the question: How complex is plan verification in hierarchical planning? This question is, by itself, of theoretical interest, as it provides further insight into the structure of hierarchical planning. Plan verification is also of practical interest whenever solutions need to be changed after the planning process and still have to fulfil all solution criteria. Such situations include plan repair or post-optimization.

Usually, plans in hierarchical planning are partially ordered. In some cases the plan that has to be verified is given as a sequence of actions instead. Consider e.g. an observed sequence of actions in plan recognition. Verifying such sequences is a slightly different problem from verifying partially ordered plans. It does not include deciding whether there is an executable linearisation of a given partially ordered plan, which has been shown to be a hard problem (Nebel and Bäckström 1994; Chapman 1987).

Lang and Zanuttini (2012) have investigated knowledge-based programs (KBP), i.e. plans containing control structures. They have shown, that in this case plan verification is Π_2^P**-complete** for KBPs without loops and **EXPSPACE-complete** in general.

A related problem is to decide if a partially ordered plan is a specialisation of a second one, i.e. it includes the same tasks, but a stricter partial order. It arises when information about the decomposition steps is available during the verification process. It is also important for planning systems in general: partial plans are the search nodes in plan-space planning (e.g. in hierarchical task network (Erol, Hendler, and Nau 1994), partial-order causal-link (POCL) (Penberthy and Weld 1992) or hybrid planning (Biundo and Schattenberg 2001)). Comparing task networks is crucial e.g. whenever visited lists are used to ensure termination of certain subclasses of HTN planning problems (Alford et al. 2012).

In this paper we show that the plan verification problem is **NP-complete**. This holds regardless of whether the plan that is to be verified is partially or totally ordered, or whether decomposition information is provided or not; and even for very simple subclasses of HTN planning problems that show e.g. neither preconditions nor effects and are very restricted in decomposition depth. HTN-like structures are also used to represent plan libraries in plan and goal recognition. We discuss what our results imply for the complexity of this task.

2 Hierarchical planning

This section provides a formal introduction to HTN planning similar to our previous definition (Höller et al. 2014) adopted from Geier and Bercher (2011). We start by describing task networks, which are partially ordered sets of tasks. A *task* is a unique identifier. A *task name* is assigned to each task giving the type of a task, e.g. *move-a-b*. The distinction between task names and tasks is necessary to allow multiple instances of a task name, i.e. an action, in a task network.

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

- *T is a finite, possibly empty, set of tasks*
- *$\prec \subseteq T \times T$ is a strict partial order on T*
- *$\alpha : T \to X$ labels every task with a task name*

TN_X is defined as the set of all task networks over the task names X. As a short-hand notation, we write $T(\mathrm{tn}) = T$, $\prec(\mathrm{tn}) = \prec$ and $\alpha(\mathrm{tn}) = \alpha$ for a task network $\mathrm{tn} = (T, \prec, \alpha)$. We define $\mathrm{tn}(x)$ to be the task network containing a single instance of x, i.e. $\mathrm{tn}(x) = (\{\circ\}, \emptyset, \{(\circ, x)\})$ and $\mathrm{tn}_\emptyset = (\emptyset, \emptyset, \emptyset)$ to be the empty task network.

Definition 2 (Isomorphic Task Network) *Two task networks* $\mathrm{tn} = (T, \prec, \alpha)$ *and* $\mathrm{tn}' = (T', \prec', \alpha')$ *are called* isomorphic*, written* $\mathrm{tn} \cong \mathrm{tn}'$*, if and only if there exists a bijection* $\sigma : T \to T'$*, such that* $\forall t, t' \in T$ *it holds that* $(t, t') \in \prec$ *if and only if* $(\sigma(t), \sigma(t')) \in \prec'$ *and* $\alpha(t) = \alpha'(\sigma(t))$.

A planning problem is defined in the following way:

Definition 3 (Planning Problem) *A* planning problem *is a 6-tuple* $\mathcal{P} = (L, C, O, M, c_I, s_I)$*, with*

- 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 primitive task name $o \in O$*, its* operator *or* action *is given by a tuple that defines its precondition and its effect, the latter in terms of an add-, and a delete list:* $(\mathit{prec}(o), \mathit{add}(o), \mathit{del}(o)) \in 2^L \times 2^L \times 2^L$.

Allowing only a single initial task name instead of an initial task network helps to make some proofs less complicated, but does not influence expressivity. Every HTN planning problem with an initial task network can easily be translated to this form by the costs of one additional compound task name and one additional method.

In HTN planning, compound tasks are repeatedly *decomposed* until all tasks are primitive. To ease notation, we define restrictions on relations, functions and task network using the bar notation.

Definition 4 (Restriction) *Let* $R \subseteq D \times D$ *be a relation,* $f : D \to V$ *a function and* tn *be a task network. Then the restrictions of* R *and* f *to some set* X *are defined by*

$$R|_X := R \cap (X \times X)$$
$$f|_X := f \cap (X \times V)$$
$$\mathrm{tn}|_X := (T(\mathrm{tn}) \cap X, \prec(\mathrm{tn})|_X, \alpha(\mathrm{tn})|_X)$$

Definition 5 (Decomposition) *A method* $m = (c, \mathrm{tn}_m) \in M$ *decomposes a task network* $\mathrm{tn}_1 = (T_1, \prec_1, \alpha_1)$ *into a 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* $\exists \mathrm{tn}' = (T', \prec', \alpha')$ *with* $\mathrm{tn}' \cong \mathrm{tn}_m$ *and* $T' \cap T_1 = \emptyset$*, where*

$$\mathrm{tn}_2 := (T'', \prec_1 \cup \prec' \cup \prec_X, \alpha_1 \cup \alpha')|_{T''} \text{ with}$$
$$T'' := (T_1 \setminus \{t\}) \cup T'$$
$$\prec_X := \{(t_1, t_2) \in T_1 \times T' \mid (t_1, t) \in \prec_1\} \cup \{(t_1, t_2) \in T' \times T_1 \mid (t, t_2) \in \prec_1\}$$

We write $\mathfrak{D}(\mathrm{tn}_1, t, m)$ *for an arbitrary but fixed task network* tn_2*, s.t.* $\mathrm{tn}_1 \xrightarrow[t,m]{} \mathrm{tn}_2$*, i.e. the canonical representative of all such task network w.r.t.* $\cong$*. We write* $\mathrm{tn}_1 \to_D^* \mathrm{tn}_2$*, if* tn_1 *can be decomposed into* tn_2 *using an arbitrary number of decompositions.*

In the common HTN problem setting, changing task networks is only possible via decomposing compound tasks. Allowing the insertion of tasks into task networks independently of task decomposition allows for more flexibility in modelling the domain and even renders the plan existence problem decidable (Geier and Bercher 2011). This setting is called HTN planning with task insertion or TIHTN. Throughout the paper we will show results for the pure HTN formalism. However, they also apply to TIHTN and its altered solution criterion.

Definition 6 (Task Insertion) *Let* $\mathrm{tn}_1 = (T_1, \prec_1, \alpha_1)$ *be a task network. Let* o *be a primitive task name; 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* $t \notin T_1$ *and* $\prec_1$ *is a strict partial order on* $T_1 \cup \{t\}$*. We write* $\mathrm{tn}_1 \to_I^* \mathrm{tn}_2$*, if* tn_2 *can be generated from* tn_1 *using an arbitrary number of insertions of primitive task names.*

Following definitions by Erol, Hendler, and Nau (1996) and Geier and Bercher (2011), we define a task network as being executable if there *exists* a linearisation of its tasks that is executable. An alternative definition could require all linearisations to be executable, as in hybrid planning (Biundo and Schattenberg 2001) and POCL planning (Penberthy and Weld 1992).

Definition 7 (Executable Task Network) *A task network* $(T, \prec, \alpha)$ *is* executable in a state $s \in 2^L$*, if and only if all its tasks are primitive and there exists a linearisation 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* $\mathit{prec}(\alpha(t_i)) \subseteq s_{i-1}$ *and* $s_i = (s_{i-1} \setminus \mathit{del}(\alpha(t_i))) \cup \mathit{add}(\alpha(t_i))$ *for all* $1 \leq i \leq n$.

Finally, we define the solutions of a planning problem $\mathcal{P}$.

Definition 8 (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) $\mathrm{tn}_I \to_D^* \mathrm{tn}_S$ *for* tn_S *being an* HTN *solution to* $\mathcal{P}$ *or*

(2') *there exists a task network* tn_B *such that* $\mathrm{tn}_I \to_D^* \mathrm{tn}_B \to_I^* \mathrm{tn}_S$ *for* tn_S *being a* TIHTN *solution to* $\mathcal{P}$.

$\mathrm{Sol}_{HTN}(\mathcal{P})$ *and* $\mathrm{Sol}_{TIHTN}(\mathcal{P})$ *denote the sets of all* HTN *and* TIHTN *solutions of* $\mathcal{P}$*, respectively.*

3 Plan Verification for Task Networks

In this section, we study the problem of HTN plan verification in its most general form. Plan verification is the question whether a given task network tn is a solution for a given planning problem $\mathcal{P}$. The following definition provides the formal decision problem.

Definition 9 (VERIFYTN) *The problem* VERIFYTN *is to decide, given a planning problem* $\mathcal{P}$ *and a task network* tn*, whether* $\mathrm{tn} \in \mathrm{Sol}_{HTN}(\mathcal{P})$ *holds.*

Please note that the task network tn is part of the input and all complexity results will refer to the combined length of the planning problem and the plan to be verified.

NP-hardness can be easily obtained for this problem. We can reduce from the problem asking whether a task network tn has an executable linearisation. One can use a planning problem with the sole method (c_I, tn) and then ask

whether tn is a solution of this problem. Since this problem has already been proven to be **NP-complete** (Nebel and Bäckström 1994, Thm. 14, 15), (Erol, Hendler, and Nau 1996, Thm. 8) the following corollary holds.

Corollary 1 VERIFYTN *is **NP-hard**.*

In the remainder of this section, we show that verifying plans can be solved in **NP**. We have provided a nondeterministic algorithm which decides whether a sequence of actions ω is a word of the formal language induced by a planning problem $\mathcal{P}$ (Höller et al. 2014, Alg. 1). Since this language contains every executable linearisation of every solution of $\mathcal{P}$, this algorithm can serve as the basis for the **NP** membership proof. However, the algorithm assumes that the HTN planning domain is in a so-called 2-normal form $\mathrm{NF}_{\geq 2}$. In this normal form the task network tn of each method (c, tn)[1] has at least size 2, i.e. $|T(\mathrm{tn})| \geq 2$. Though the given construction preserves the problem's set of solutions, it leads to exponentially many new decomposition methods. It is unknown whether it is possible to find an equivalent planning problem of sub-exponential size. This makes the construction of the $\mathrm{NF}_{\geq 2}$ unsuitable for an **NP**-membership proof. We showed that the algorithm is linear space bounded, which does only imply **PSPACE** membership but not **NP** membership. Here we enhance the algorithm s.t. it can verify plans for arbitrary HTNs and show that the resulting algorithm is still in **NP**.

The given proof would work with less effort if methods decomposing into empty task networks (henceforth called ε-methods) are forbidden in the input. However, this would restrict the options available for domain modellers.

To overcome the restrictions of the previous algorithm, we use the notions of *possibly empty task* Π_ε and of the *unit reachability function* ρ_1. These two represent the transformation of a planning domain into $\mathrm{NF}_{\geq 2}$ in a compressed way. We define both of them and show that they can be computed in polynomial time. Please be aware that ρ_1 is only necessary for the construction of Π_ε and is not used later on.

Definition 10 *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem. We define the set of deletable abstract tasks Π_ε as*

$$\Pi_\varepsilon = \{a \mid \mathrm{tn}(a) \rightarrow_D^* \mathrm{tn}_\emptyset\}$$

We define the unit reachability function $\rho_1 : C \rightarrow 2^{C \cup O}$ as

$$\rho_1(c) = \{a \mid \mathrm{tn}(c) \rightarrow_D^* \mathrm{tn}(a)\}$$

The computation of Π_ε and ρ_1 is necessarily intertwined. As such we compute them inductively. The base-cases are:

$$\Pi_\varepsilon^0 = \{a \mid (a, \mathrm{tn}) \in M \text{ and } |T(\mathrm{tn})| = 0\}$$
$$\rho_1^0(c) = \{c\} \cup \{a \mid (c, \mathrm{tn}) \in M \text{ and } T(\mathrm{tn}) = \{t\} \text{ with } \alpha(t) = a\}$$

Subsequently the inductive step is given by

$$\Pi_\varepsilon^n = \Pi_\varepsilon^{n-1} \cup \{a \mid (a, \mathrm{tn}) \in M \text{ and } \forall t \in T(\mathrm{tn}) : \rho_1^{n-1}(t) \cap \Pi_\varepsilon^{n-1} \neq \emptyset\}$$

$$\rho_1^n(c) = \rho_1^{n-1}(c) \cup \bigcup_{a \in \rho_1^{n-1}(c)} \rho_1^{n-1}(a) \cup \{a \mid (c, \mathrm{tn}) \in M \text{ and } \exists t \in T(\mathrm{tn}) : \alpha(\mathrm{tn})(t) = a \text{ and } \forall t' \in T(\mathrm{tn}) \setminus \{t\} : \alpha(\mathrm{tn})(t') \in \Pi_\varepsilon^{n-1}\}$$

Clearly, $\Pi_\varepsilon = \Pi_\varepsilon^\infty$ and $\rho_1 = \rho_1^\infty$ holds. The iteration can be aborted if Π_ε^n and ρ_1^n have not changed any more. Since both are bounded in size, at most $\gamma = |C| + |C|(|C| + |O|)$ steps need to be performed. Each step takes by definition an effort of at most $|M|\delta\gamma$, where $\delta = \max_{(c,\mathrm{tn}) \in M} |T(\mathrm{tn})|$. Thus the computation of Π_ε and ρ_1 takes at most $|M|\delta\gamma^2$ steps, which is still polynomial.

Next we can define the ε-extended planning problem containing "short-cut" decompositions for all tasks in Π_ε.

Definition 11 *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem. Then the problem $\mathcal{P}_\varepsilon = (L, C, O, M \cup \{(c, \mathrm{tn}_\emptyset) \mid c \in \Pi_\varepsilon\}, c_I, s_I)$ is the ε-extended planning problem.*

At this point it should be obvious that $\mathrm{Sol}_{HTN}(\mathcal{P}) = \mathrm{Sol}_{HTN}(\mathcal{P}_\varepsilon)$ holds, the only difference between the problems being that decomposing c_I may take fewer method applications due to the new ε-methods. Prior to the **NP**-membership proof, we show the following lemma providing an upper bound to the number of decompositions necessary to reach a certain task network starting with $\mathrm{tn}(c_I)$.

Lemma 1 *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be an ε-extended planning problem and tn_s a non-empty task network s.t. $\mathrm{tn}(c_I) \rightarrow_D^* \mathrm{tn}_s$. Then $k < |T(\mathrm{tn}_s)|(|C| + 1)$ task networks $\mathrm{tn}_1, \ldots, \mathrm{tn}_k$ exist such that $\mathrm{tn}(c_I) = \mathrm{tn}_0 \rightarrow_D^* \mathrm{tn}_1 \rightarrow_D^* \ldots \rightarrow_D^* \mathrm{tn}_k \rightarrow_D^* \mathrm{tn}_{k+1} = \mathrm{tn}_s$ and $\forall i : |T(\mathrm{tn}_i)| \leq |T(\mathrm{tn}_{i+1})|$ and decomposing any tn_i into tn_{i+1} uses exactly one decomposition method $m_i = (c, \mathrm{tn})$ with $T(\mathrm{tn}) \geq 1$ followed by an arbitrary number of ε-methods $\tilde{m}_i$ applied only to the tasks inserted by m_i.*

Proof: Since $\mathrm{tn}(c_I) \rightarrow_D^* \mathrm{tn}_s$ there exists some tn_i, s.t. $\mathrm{tn}(c_I) \rightarrow_D^* \mathrm{tn}_1 \rightarrow_D^* \ldots \rightarrow_D^* \mathrm{tn}_k \rightarrow_D^* \mathrm{tn}_s$ for some k where each tn_i is decomposed into tn_{i+1} by using a single decomposition method $m_i = (c, \mathrm{tn})$ with $T(\mathrm{tn}) \geq 1$ followed by an arbitrary number of ε-methods. In addition, we can require w.l.o.g. that decompositions using ε-methods follow immediately after the method creating the task they delete, since we can re-arrange their order. Thus the second required property holds. We need to show that there is also a sequence of task networks with monotonic size.

Suppose there is a task network tn_i, s.t. $|T(\mathrm{tn}_i)| > |T(\mathrm{tn}_{i+1})|$. To achieve this, ε-methods must have been applied. Following the above assumption, these ε-methods are only applied to the tasks introduced by m_i. Since $|T(\mathrm{tn}_i)| > |T(\mathrm{tn}_{i+1})|$, all these newly introduced tasks must have been decomposed into the empty task network, i.e. erased. Thus the task that was decomposed by m_i is a member of Π_ε. It could have been deleted directly after the decomposition step which created it. In the respective sequence of decompositions $|T(\mathrm{tn}_i)| = |T(\mathrm{tn}_{i+1})|$ holds. Applying this inductively, we obtain $|T(\mathrm{tn}_i)| \leq |T(\mathrm{tn}_{i+1})|$ for all i.

What remains is to show is that k is bounded. W.l.o.g. we can assume that *unit methods* (methods for which $|T| = 1$) are applied immediately before applying another method to the newly introduced task. Otherwise this renaming could

[1] The only exception from this rule is the initial task c_I, that in turn must not be contained in any methods task network.

be postponed. If $|T(\mathrm{tn}_i)| = |T(\mathrm{tn}_{i+1})|$ holds for a step in the resulting sequence, then it is either a unit method, or a regular method generating a task network of size s where $s-1$ ε-methods are applied immediately after the decomposition. For any sequence $\mathrm{tn}_i \rightarrow_D^* \ldots \mathrm{tn}_{i+l}$ where $|T(\mathrm{tn}_i)| = \cdots = |T(\mathrm{tn}_{i+l})|$ only a single task $t \in T(\mathrm{tn}_i)$ is changed into an other task in every step. If $l > |C|+1$[2] there is at least one task network which is repeated. Hence there is an equivalent but shorter sequence.

As the number of decompositions, not changing the size of the task network, is limited by $|C|+1$, the total number of task networks is limited by $|T(\mathrm{tn}_{sol})|(|C|+1)$. □

Having this property at hand we can give an algorithm that decides whether a task network tn is a solution of a given planning problem $\mathcal{P}$. The main idea of the algorithm is to non-deterministically choose decompositions and apply them to the current task network until it is equivalent to the task network provided in the input. By applying decomposition methods of the ε-reduced planning problem to each intermediate network, we can use Lemma 1 to give an upper bound on the number of applied decompositions.

Theorem 1 VERIFYTN *is in* ***NP****.*

Proof: The non-deterministic algorithm given in Algorithm 1 decides whether $\mathrm{tn}_s \in \mathrm{Sol}_{HTN}(\mathcal{P})$ holds in a given planning problem $\mathcal{P}$. Each step in the algorithm has at most polynomial complexity. If the algorithm accepts the input, it has found an executable linearisation of tn_s and has constructed a task network isomorphic to tn_s by decomposing $\mathrm{tn}(c_I)$ and thus has shown that $\mathrm{tn}_s \in \mathrm{Sol}_{HTN}(\mathcal{P})$ holds.

Provided $\mathrm{tn}(c_I) \rightarrow_D^* \mathrm{tn}_s$ holds, either $\mathrm{tn}_s = \mathrm{tn}_\emptyset$, which is handled by line 2, or tn_s is not empty. Then we know by using Lemma 1 that there is a decomposition of $\mathrm{tn}(c_I)$ into tn_s which has at most $|T(\mathrm{tn}_s)|(|C|+1)$ intermediate task networks tn_i s.t. tn_{i+1} is obtained from tn_i by applying a single decomposition method m_i with a task network of size $k \geq 1$, followed by at most $k-1$ applications of ε-methods from the ε-reduced planning domain. Our algorithm only computes these intermediate steps, i.e. it considers in the i^{th} iteration of the loop the task network tn_i, starting with $\mathrm{tn}_0 = \mathrm{tn}(c)$. It then non-deterministically selects a task and a method m_i to apply to it (lines 8 and 9) and a set of ε-methods, represented as a set E of tasks to be deleted (line 10). These methods are immediately applied to the method itself by deleting the respective tasks (line 11). The newly obtained method is then applied to decompose the just selected task in tn_i. Since a proper subset of tasks of the method's task network is selected, at most $|T|-1$ ε-methods are applied and thus the size of tn never decreases. Hence there is a set of choices for our non-deterministic algorithm which leads to the task network tn_s in at most $|T(\mathrm{tn}_s)|(|C|+1)$ iterations of the loop. □

Combining the two results from Corollary 1 and Theorem 1 we obtain that VERIFYTN is **NP-complete**.

Corollary 2 VERIFYTN *is* ***NP-complete****.*

[2] all compound tasks and one primitive task need to be traversed

```
1  function verify(P = (L, C, O, M, c_I, s_I), tn_s)
2      if |T(tn_s)| = 0 then return c_I ∈ Π_ε;
3      if ∃t ∈ T(tn_s) : α(t) ∉ O then return failure;
4      Guess a linearisation ω of tn_s
5      if ω is not executable then return failure;
6      tn := tn(c_I); i := 0
7      while |T(tn)| ≤ |T(tn_s)| ∧ i ≤ |T(tn_s)|(|C|+1) do
8          Choose a task d ∈ T(tn)
9          Choose m = (α(d), (T, ≺, α)) ∈ M
10         Choose subset E ⊆ T s.t. ∀x ∈ E : α(c) ∈ Π_ε
11         tn^E := (T, ≺, α)|_{T\D}
12         tn := 𝔇(tn, d, (α(d), tn^E))
13         i := i + 1
14     if |T(tn)| ≠ |T(tn_s)| then return failure;
15     Choose a bijection φ : T(tn) → T(tn_s)
16     if ∃t ∈ T(tn) : α(tn)(t) ≠ α(tn_s)(φ(t)) then
17         return failure
18     if φ(≺(tn)) ≠ ≺(tn_s) then return failure;
19     return success
```

Algorithm 1: Non-deterministic polynomial-time Algorithm, deciding whether tn is a solution of $\mathcal{P}$

4 Plan Verification for Task Sequences

In the previous section, we have shown that plan verification is **NP-complete** if a task network has to be verified. The presented reason for **NP-hardness** was the task of finding a executable linearisation of that task network. Consequently, one might pose the question whether the verification problem is easier if such a linearisation is already given. In many real world-scenarios we are provided with such a sequence, e.g. in plan and goal recognition. Unfortunately, this section shows that having such a sequence at hand does not make the problem easier – it remains **NP-complete**.

Definition 12 (VERIFYSEQ) *The problem* VERIFYSEQ *is to decide, given an HTN planning domain $\mathcal{P}$ and a task sequence ω, whether $\exists \mathrm{tn} \in \mathrm{Sol}_{HTN}(\mathcal{P})$ s.t. ω is an executable linearisation of* tn*.*

In addition to **NP-completeness** of this general problem, we will also show it for many severely restricted domains. However, we want to mention that there is a restriction on planning problems making plan verification tractable.

Lemma 2 VERIFYSEQ *is* ***P-complete*** *for* HTN *planning problems with totally ordered methods.*

Proof: These HTN planning problems can easily be transformed into context-free grammars (Höller et al. 2014). Thus VERIFYTN is equivalent to the word problem of such grammars and thus **P-complete** (Jones and Laaser 1974). □

Please note that this lemma only applies to our HTN-definition allowing a single initial task, while in Erol's definition the initial task network must also be totally ordered.

The **NP** membership of the VERIFYSEQ problem for partially ordered planning problems is a direct corollary from the proof for Theorem 1. The presented algorithm checks whether the task network tn_s is isomorphic to a task network

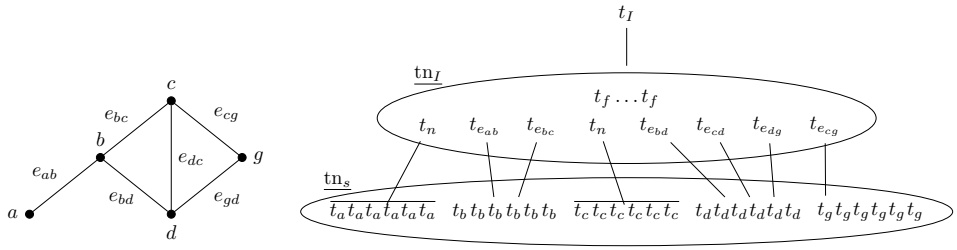

Figure 1: Illustrative example for the reduction of vertex cover to VERIFYSEQ. The graph on the left is encoded as HTN planning problem for cover of size 3. t_I is decomposed into tn_I (inside the upper ellipse). Several other decompositions, each given by a connecting edge, refine it to ω (the bottom ellipse). The t_n tasks occupy all instances of t_a and t_c, so that these tasks can not be included in the cover. The t_e tasks ensure that one end of each edge is included in the cover. The t_f tasks generate the remaining tasks in ω.

tn obtained from the solution. Instead it could test whether the input task network, in our case being a sequence, is a specialization of tn by altering line 18 of the algorithm.

Corollary 3 VERIFYSEQ *is in* ***NP****.*

To show **NP-hardness** for the VERIFYSEQ problem, we adapt a **NP-hardness** proof for parsing ID/LP grammars provided by Barton (1985). He reduces the **NP-complete** vertex cover (VC) problem (Karp 1972) to this problem.

Definition 13 (VERTEXCOVER) *The* VERTEXCOVER *problem is to decide, given a graph $G = (V, E)$ and a number k, whether it is possible to find a subset of nodes $S \subseteq V$ with $|S| \leq k$, the vertex cover, so that each of the edges is adjacent to a node in the set S, i.e. $\forall e \in E : e \cap S \neq \emptyset$.*

We start by giving the proof for full HTN planning and later discuss which restrictions can be imposed while preserving **NP-hardness**.

Theorem 2 VERIFYSEQ *is* ***NP-hard****.*

Proof: Let $G = (V, E)$ be a graph and k a number. If $k \geq |V|$, such a vertex cover obviously exists, so let $k < |V|$. Now we need to construct a planning problem $\mathcal{P}$ and a sequence of tasks ω, such that

$$\exists \mathrm{tn}_s : \mathrm{tn}_s \in \mathrm{Sol}_{HTN}(\mathcal{P}) \text{ and } \omega \text{ is a linearisation of } \mathrm{tn}_s \\ \Leftrightarrow G \text{ has a VC of size } \leq k$$

We define the planning problem $\mathcal{P} = (\emptyset, C, O, M, t_I, \emptyset)$ corresponding to the graph G as follows.

The planning domain contains a primitive task t_v for each node in V. They have neither preconditions nor effects.

$$O = \{t_v \mid v \in V\}$$

For each edge $e \in E$, a compound task t_e is introduced. All edges have to be adjacent to (at least) one node in the VC. There are two methods for each task t_e, decomposing it in a task that represents the node at one end of the edge. Applying one of them chooses the node in the VC.

$$M_E = \{(t_e, \mathrm{tn}(t_v)) \mid e \in E, v \in e\}$$

A vertex cover is chosen by decomposing a task network containing all t_e into primitive tasks. The remaining parts of the planning problem and the input sequence ω ensure that no more than k nodes are included in the VC. If the vertex cover has size $\leq k$, there are at least $|V| - k$ vertices that are not included in the VC. The compound task t_n (non-chosen) represents one of these nodes. Since we do not know which of the nodes will not be members of the VC, the domain contains a decomposition method $m_v^{t_n}$ for each $v \in V$. Each method $m_v^{t_n}$ decomposes t_n in $|E|$ unordered copies of the primitive task t_v. The input sequence ω contains each task that represents a node $|E|$ times. This enforces that only vertexes in the cover can be chosen when decomposing a t_e.

$$M_N = \{(t_n, ([|E|]^3, \emptyset, \{(n, t_v) \mid n \in [|E|]\})) \mid v \in V\}$$

We are not finished defining the planning problem, but before we explain the remaining part, let's have a look at Figure 1. The left side shows a graph with 5 vertices and 6 edges. Assume we are asked for a 3-VC. The planning problem is given in Figure 1(b). The initial task produces $5-3=2$ times t_n as well as the 6 edge tasks t_e.

Now only generating the remaining instances of tasks representing the vertices included in the cover is left. The compound task t_f provides these tasks. t_f has a method for each $v \in V$, decomposing it into a single primitive task t_v.

$$M_F = \{(t_f, \mathrm{tn}(t_v)) \mid v \in V\}$$

Finally we define the method decomposing the initial task t_I by $m^{t_I} = (t_I, \mathrm{tn}_I)$. The task network tn_I contains one instance of each t_e, $|V| - k$ times $t_n$and $(k-1) \cdot |E|$ times t_f. The latter ensures that it is possible to obtain $|E|$ instances of each vertex in the cover. The task network is totally unordered. m^{t_I} is defined by

$$m^{t_I} = (t_I, (T_I, \emptyset, \alpha_I)) \text{ where}$$
$$T_I = \{\circ_e^t \mid e \in E\} \cup \{\circ_1^n, \dots, \circ_{|V|-k}^n, \circ_1^f, \dots, \circ_{(k-1)|E|}^f\}$$
$$\alpha_I = \{(\circ_e^t, t_e) \mid e \in E\} \cup \{(\circ_i^n, t_n) \mid i \in [|V| - k]\} \\ \cup \{(\circ_i^f, t_f) \mid i \in [(k-1)|E|]\}$$

[3] We use $[n]$ to abbreviate the set $\{1, \dots, n\}$

With the input sequence $\omega = t_{v_1}^{|E|} \dots t_{v_{|V|}}^{|E|}$, the planning problem $\mathcal{P}$ is completed by

$$C = \{t_n, t_f, t_I\} \cup \{t_e \mid e \in E\}$$
$$M = \{m^{t_I}\} \cup M_E \cup M_N \cup M_F$$

$\Rightarrow$: Let tn_s be a solution of $\mathcal{P}$ s.t. ω is a linearisation of tn_s. Since tn_s is a solution, it does not contain any compound tasks. Thus it does not contain any t_e tasks, each such task in tn_I has been decomposed into a primitive task t_v where $v \in e$. We will show that the set of all these v is a vertex cover of G with size at most k. It is obvious that this set is a vertex cover: for every edge one of its nodes was chosen.

Every decomposition for a compound task t_n generates exactly $|E|$ instances of some primitive task t_v for some $v \in V$. Since ω contains only $|E|$ instances of each t_v, different decomposition methods must have been chosen for each t_n. Let $\overline{C}$ be the set of all types of primitive tasks obtained by decomposing t_n tasks. Clearly $|\overline{C}| = |V| - k$ holds. Suppose any t_e was decomposed into a member t_v of $\overline{C}$, then ω must contain at least $|E| + 1$ instances of t_v, which is a contradiction. Hence the set of all primitive tasks generated by decomposing t_e tasks is disjoint from $\overline{C}$ and thus is a VC with size of at most k.

$\Leftarrow$: Let C be a vertex cover of G of size k. We describe how $\mathrm{tn}(t_I)$ can be decomposed into tn_s having ω as an executable linearisation. We will choose tn_s as the task network that contains each task t_v $|E|$ times and is totally unordered. Clearly tn_s has ω as a possible linearisation and ω is trivially executable. Initially t_I will be decomposed using m^{t_I} into an t_e task for each $e \in E$, $|V| - k$ instances of t_n and $(k-1) \cdot |E|$ instances of t_f.

For each task t_n we choose a different node $v \in V \setminus C$. This is possible as $|V \setminus C| = |V| - k$. Each t_n is decomposed using the method $m_v^{t_n}$, i.e. into $|E|$ instances of V.

For each t_e task at least one of the nodes of e is a member of C, let that node be t_v[4]. t_e is decomposed using the method inserting t_v. These decompositions will generate $|E|$ instances of nodes $t_v \in C$. No node will be generated more than $|E|$ times. For each node $t_v \in C$, $|E|$ instances must be created to obtain the task network tn_s. Using the $(k-1)|E|$ instances of t_f created by m^{t_I}, the missing instances of nodes $t_v \in C$ can be obtained. After applying these methods, tn_I has been decomposed into $|E|$ copies of each node $v \in V$. None of the used methods introduces any ordering constraint and no preconditions are present. We have obtained the task network tn_s by decomposing tn_I, which concludes the proof. □

Combining the two results from Corollary 3 and Theorem 2 we obtain that VERIFYSEQ is **NP-complete**, too.

Corollary 4 VERIFYSEQ *is* ***NP-complete****.*

The given proof does not only apply to the class of full HTN planning problems, but also to much more restricted classes. The planning problem $\mathcal{P}$ defined in the proof does neither contain preconditions nor effects, all its methods are totally unordered, i.e. they don't contain any ordering, its decomposition methods are not recursive and the maximal "depth" of decompositions is 2. We denote these classes – in the same order – as $HTN_{0\ eff}^{0\ pre}$, $HTN_{unordered}$, HTN_{acyc} and HTN_{2dec}. Please be aware that the first decomposition step is only necessary because we allowed only a single initial task instead of an initial task network in our HTN definition. If an initial task network would be allowed, only one decomposition step is necessary for the proof, i.e. the problem is already **NP-hard** when compound tasks are solely allowed in the initial task network (but not in methods' task networks).

Corollary 5 VERIFYSEQ *for the classes* $HTN_{0\ eff}^{0\ pre}$, $HTN_{unordered}$, HTN_{acyc} *and* HTN_{2dec} *and any of their intersections is* ***NP-complete****.*

Furthermore the presented proof does, with minimal modifications, also hold for the VERIFYTN problem. One can conclude that VERIFYTN is **NP-complete**, even if neither preconditions nor effects are allowed. In this case Corollary 1 does not apply.

Corollary 6 VERIFYTN *for* $HTN_{0\ eff}^{0\ pre}$ *is* ***NP-complete****.*

Our proofs for Theorem 1 and Theorem 2 can easily be adapted for HTN planning with task insertion ($TIHTN$).

Corollary 7 VERIFYSEQ *for* $TIHTN$ *is* ***NP-complete****.*

5 Compatibility of Plans

So far we assumed that we do not have any information about the decompositions which lead to the plan to be verified. Clearly, Corollary 1 still holds if the list of decompositions is provided, since it is only based on the need to show that an executable linearisation of the input task network exists. However, even if executability of the task networks that have to be checked can be assumed, and thus does not have to be tested, the remaining problem is still **NP-hard**.

The remaining question is to decide whether the task network is "compatible" with the one obtained by applying the decompositions. Informally compatibility means that the set of task names are identical and one of the networks has a more restrictive partial order $\prec$. Albeit this task might seem trivial, it is not. In general, the identifier of the tasks in tn, given by $T(\mathrm{tn})$, are different from those of the task network resulting from applying the given decompositions. As a consequence, tasks with the same name in one task network might be mapped to any such task in the other one.

Checking task network compatibility is also of interest for planners in general. As task networks define partial plans, they represent the search nodes in systems that use plan-space search (e.g. in HTN, POCL or hybrid planning). Therefore comparing task networks is crucial e.g. whenever visited lists are used. Recent work has shown that using visit lists ensures termination of certain subclasses of HTN planning (Alford et al. 2012). Each newly generated task network tn_N is compared to a list of already visited task networks tn_i. The easiest variant of this test is to test isomorphism between tn_N and each tn_i. It is easy to show that this

[4] If both nodes are members of C either may be chosen as t_v

test is graph-isomorphism complete. Despite it is reasonable to assume that **GI-complete** problems are not in **P**, they are tractable in practical cases. This is especially true for the task network isomorphism problem, since usually only a few instances of the same task are contained in a plan, making computation easier. If task compatibility is checked instead of isomorphism the loop-detection procedure is tighter, i.e. it reduces the search space even more. Hence studying its complexity is important for planning systems. We define compatibility of two task networks and the respective decision problem.

Definition 14 (Compatibility of Task Networks) *Let* tn_1 *and* tn_2 *be task networks.* tn_1 *is compatible with* tn_2*, written as* $\mathrm{tn}_1 \rhd \mathrm{tn}_2$*, if and only if there is a task network* $\mathrm{tn}'_1 = (T(\mathrm{tn}_1), \prec'_1, \alpha(\mathrm{tn}_1))$ *with* $\prec'_1 \subseteq \prec(\mathrm{tn}_1)$ *s.t.* $\mathrm{tn}'_1 \cong \mathrm{tn}_2$.

Definition 15 (PLANCOMPATIBILITY) *The* PLANCOMPATIBILITY *problem is to decide, given two task networks* tn_1 *and* tn_2*, whether* $\mathrm{tn}_1 \rhd \mathrm{tn}_2$ *holds.*

We prove that the PLANCOMPATIBILITY problem is **NP-complete** by reducing the **NP-complete** subgraph isomorphism problem to PLANCOMPATIBILITY (Cook 1971).

Definition 16 (SUBGRAPHISO) *The subgraph isomorphism problem (*SUBGRAPHISO*) is to decide, given two graphs* $G = (V_G, E_G)$ *and* $H = (V_H, E_H)$*, whether* G *has a subgraph*[5] $G^* = (V^*_G, E^*_G)$ *which is isomorph to* H.

Theorem 3 PLANCOMPATIBILITY *is* ***NP-complete***.

Proof: Membership: Let tn_1 and tn_2 be task networks. If $|T(\mathrm{tn}_1)| \neq |T(\mathrm{tn}_2)|$ the algorithm returns false, since no such tn'_1 can exist. Else, we can non-deterministically guess a subset $\prec'_1 \subseteq \prec(\mathrm{tn}_1)$ and a bijection $\phi : T(\mathrm{tn}_1) \to T(\mathrm{tn}_2)$. To test whether tn_1 is isomorphic to tn_2 under this bijection is polynomial, since only $\mathcal{O}(|T(\mathrm{tn}_1)|^2)$ tests are necessary.

Hardness: Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two arbitrary graphs. We need to construct two task networks tn_1 and tn_2 s.t. tn_1 is a specialization of tn_2 if and only if G has a subgraph isomorphic to H. The construction and the idea of the reduction is illustrated in Figure 2.

The task network tn_1 will represent the structure of G. It contains a task for every vertex and edge of G. All tasks are labelled with the same name $\circ$. Each task, representing an edge is ordered after the two tasks representing its nodes. Formally we define the task network tn_1 by

$$\mathrm{tn}_1 = (V_G \cup E_G, \{(v, e) \mid e \in E_G, v \in e\}, \{(t, \circ) \mid t \in T(\mathrm{tn}_1)\})$$

The task network tn_2 encodes the graph H. The graph H can have less vertices and edges than G, however the number of tasks in tn_2 must be equal to the number in tn_1. If not, these two task networks can not be compatible. Thus we add an appropriate number of isolated tasks to the task network tn_2. Apart from that the construction is the same, thus the task network tn_2 is defined by

$$\mathrm{tn}_2 = (V_H \cup E_H \cup \{f_i \mid i \in [|V_G \cup E_G| - |V_H \cup E_H|]\}, \{(v, e) \mid e \in E_H, v \in e\}, \{(t, \circ) \mid t \in T(\mathrm{tn}_2)\})$$

[5] A Graph $I = (V_I, E_I)$ is a subgraph of a graph $J = (V_J, E_J)$ if and only if $V_I \subseteq V_J$ and $E_I \subseteq E_J \cap (V_I \times V_I)$

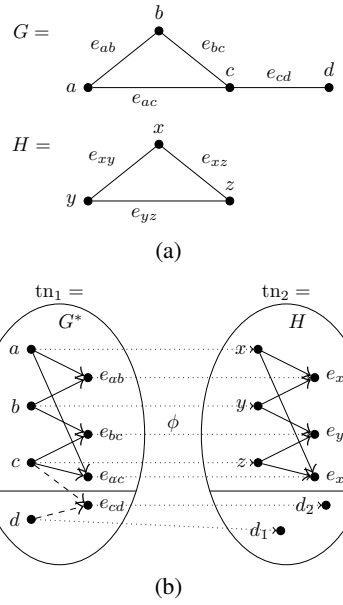

Figure 2: Illustrative example for the reduction of SUBGRAPHISO to PLANCOMPATIBILITY. The two graphs are given in 2(a). The corresponding task networks in 2(b). Be aware the initially different cardinality of the node sets and that the encoding results in task networks that contain no nodes that are ordered transitively.

$\Rightarrow$: Let tn_1 be compatible with tn_2. We need to show that there is a subgraph of G isomorphic to H. Since tn_1 is compatible to tn_2, a task network $\mathrm{tn}'_1 = (T(\mathrm{tn}_1), \prec'_1, \alpha(\mathrm{tn}_1))$ where $\prec'_1 \subseteq \prec(\mathrm{tn}_1)$ exists being isomorphic to tn_2. Let $\phi : T(\mathrm{tn}'_1) \to T(\mathrm{tn}_2)$ be the respective isomorphism.

We define $V^*_G = \{v \mid v \in V_G, \phi(v) \in V_H\}$, the set of vertices of G (tasks of tn_1) mapped by the isomorphism to vertices of H (tasks of tn_2). Similarly we define the set $E^*_G = \{e \mid e \in E_G, \phi(e) \in E_H\}$. We will show that the subgraph $G^* = (V^*_G, E^*_G)$ is isomorph to H and that $\psi = \phi|_{V^*_G}$ is the respective isomorphism.

Let $v_1, v_2 \in V^*_G$ be two arbitrary vertices of G^*. Then $\psi(v_1) \in V_H$ and $\psi(v_2) \in V_H$ holds.

Case 1: $e = v_1 v_2 \in E^*_G$. Thus tn_1 contains the ordering constraints $v_1 \prec e = v_1 v_2$ and $v_2 \prec e$. Since ϕ is an isomorphism, the task network tn_2 contains the ordering constraints $\phi(v_1) \prec \phi(e)$. Hence the edge $\phi(e)$ in tn_2 is ordered after two other tasks and is by construction a member of E_H.

Case 2: $v_1 v_2 \notin E^*_G$. Assume that $e = \psi(v_1)\psi(v_2) \in E_H$. Then tn_2 contains e with two predecessors, $\psi(v_1)$ and $\psi(v_2)$, in the partial order. Since ψ is an isomorphism, E^*_G contains the edge $\phi^{-1}(\psi(v1))\phi^{-1}(\psi(v2))$. As ψ and ϕ are equal on V^*_G this is edge $v_1 v_2$ which was assumed to not exist.

$\Leftarrow$: Let G have a subgraph G^* which is isomorphic to H and let $\psi : V^*_G \to V_H$ be that isomorphism. We show that network $\mathrm{tn}'_1 = (T(\mathrm{tn}_1), \prec(\mathrm{tn}_1)|_{V^*_G \cup E^*_G})$ is isomorphic to

tn_2. First we define $\tilde{\phi} : V_G^* \cup E_G^* \to V_H \cup E_H$ by

$$\tilde{\phi}(x) := \begin{cases} \psi(x) & \text{, iff } x \in V_G^* \\ \psi(v_1)\psi(v_2) & \text{, iff } x = v_1 v_2 \in E_G^* \end{cases}$$

Let further be ϕ an arbitrary bijective extension of $\tilde{\phi}$ onto $T(\mathrm{tn}_1') \to T(\mathrm{tn}_2)$. We use ϕ as the isomorphism between tn_1 and tn_2. Since both, tn_1 and tn_2, map all tasks to the same task name, only the ordering constraints must be checked.

Case 1: $t_1 \prec t_2$ holds in tn_1'. Then $t_1 \in V_G^*$, $t_2 \in E_G^*$ and t_1 is adjacent via t_2 to some vertex v in G^*, i.e. $t_2 = t_1 v$. Since ψ is an isomorphism, the edge $\phi(t_2) = \psi(t_1)\psi(v)$ must exist in H. Thus $\phi(t_1) \prec \phi(t_2)$ holds in tn_2.

Case 2: $t_1 \not\prec t_2$ holds in tn_1'. If either $t_1 \notin V_G^* \cup E_G^*$ or $t_2 \notin V_G^* \cup E_G^*$ it was mapped by ϕ to a task $t \in T(\mathrm{tn}_2) \setminus (V_H \cup E_H)$ and thus has no ordering to any other task in tn_2. If either $t_1, t_2 \in V_G^*$ or $t_1, t_2 \in E_G^*$ then by definition there can not be an ordering between $\phi(t_1)$ and $\phi(t_2)$ in tn_2. Let w.l.o.g. be $t_1 \in V_G^*$ and $t_2 = v_1 v_2 \in E_G^*$. Suppose the (only possible) ordering $\phi(t_1) \prec \phi(t_2) = \psi(v_1)\psi(v_2)$ would hold in tn_2. Then t_1 is either v_1 or v_2, let it w.l.o.g. be v_1. The vertex $\psi(t_2)$ would be connected to $\psi(v_2)$ in H. Since ψ is an isomorphism the edge $v_1 v_2$ would also exist in G^* and hence also the ordering $v_1 = t_1 \prec v_1 v_2 = t_2$ in tn_1'. □

6 Plan Recognition

Here we give a brief excursion on what our results mean for plan and goal recognition. This is motivated not only by situations where HTN plans have to be recognized, but also because HTN-like formalisms are used to define plan libraries for plan recognition.

> "Much of the past work in plan recognition has at least tacitly been based on simple hierarchical task networks ... as the representation for plans." (Geib 2004, p. 1)

Geib (2004) examines how the set of possible explanations for a sequence of observations evolves when new observations are added. The plan library is given as an HTN-style AND/OR graph and an explanation is a selection of choices at OR-nodes that result in a plan that may start with the given prefix. He thereby restricts his analysis on non-recursive libraries but allows for more than one top-level-goal. He identifies library properties that cause exponential increase in the number of explanations.

This definition based on explanation sets makes it difficult to give a hardness proof for plan recognition. Thus we give an alternative definition as a decision problem. It is easy to see that the general problem is **semi-decidable**. By using the results of the previous sections, we show that it is **NP-hard** even with additional assurances. We do allow for arbitrary HTN planning problems do define the library of valid plans.

Definition 17 (PLANREG) *The plan recognition problem is the problem of deciding, based on a planning problem $\mathcal{P}$ and a sequence of observations o, whether there is a plan solving $\mathcal{P}$ where o is the prefix of a valid linearisation.*

Though PLANREG and VERIFYTN are related problems, only a prefix of a plan is known in plan recognition. Thus the VERIFYTN problem can be seen as the special case of plan recognition where (1) all actions of the plan have been seen and (2) this information is given. Without this additional assurance, the given problem is semi-decidable, because it includes solving an HTN planning problem.

Theorem 4 PLANREG *is* ***semi-decidable***.

Proof: Assume the problem is decidable, then there exists a function $f(\mathcal{P}, o) \to \{\top, \bot\}$ that decides for an arbitrary planning problem $\mathcal{P}$ and an arbitrary sequence of observations o, if $\mathcal{P}$ has a solution that has a linearisation with the prefix o. We define a function $h(\mathcal{P}) := f(\mathcal{P}, \varepsilon)$ that returns whether there is a solution for an arbitrary HTN planning problem, which has been shown to be semi-decidable. Our assumption that the problem is decidable must be wrong.

What remains is to show that it is semi-decidable. If there exists a solution for the planning problem that has a linearisation with the given prefix, it can be found by non-deterministically decomposing the initial task until all tasks are primitive. Afterwards a linearisation is guessed and the prefix is checked. □

Theorem 5 *Given the assurance that all actions of the entire plan have been seen,* PLANREG *is* ***NP-complete***

Proof: Having this assurance, PLANREG is equivalent to VERIFYSEQ and thus **NP-complete**. □

7 Conclusion

In this paper we studied the problem of plan verification. Verifying plans for classical planning problems is trivially in **P**. We showed that, due to the additional requirements on solutions in HTN planning, this test becomes **NP-complete**, even if a witness for executability is provided. Furthermore, it remains **NP-complete** if the structure of the HTN planning problem is severely restricted. Such restrictions include the absence of preconditions and effects, of ordering in methods, of recursion in the decomposition hierarchy, and the restriction of the depth of this hierarchy. In case the applied decompositions are provided, plan compatibility has to be tested, which is also **NP-complete**.

Plan verification is also of practical interest, as it naturally occurs whenever plans have to be modified or visited lists for plan-space planners are implemented. Finally we discussed implications of our results to the complexity of plan and goal recognition.

Acknowledgments

We want to thank Pascal Bercher and the reviewers for their help to improve the paper. This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

Alford, R.; Bercher, P.; and Aha, D. 2015. Tight bounds for HTN planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015)*. AAAI Press.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2012. HTN problem spaces: Structure, algorithms, termination. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, (SoCS 2012)*, 2–9.

Barton, G. E. 1985. On the complexity of ID/LP Parsing. *Computational Linguistics* 11(4):205–218.

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In *Proceedings of the 6th European Conference on Planning (ECP 2001)*, 157–168.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC 1971)*, 151–158.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the International Conference on AI Planning & Scheduling (AIPS 1994)*, 249–254.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence* 18(1):69–93.

Geib, C. W. 2004. Assessing the complexity of plan recognition. In *Proceedings of the 19th National Conference on Artifical Intelligence (AAAI 2004)*, 507–512.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1955–1961.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 447–452.

Jones, N. D., and Laaser, W. T. 1974. Complete problems for deterministic polynomial time. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing (STOC 1974)*, 40–46.

Karp, R. M. 1972. Reducibility among Combinatorial Problems. In Miller, R. E.; Thatcher, J. W.; and Bohlinger, J. D., eds., *Complexity of Computer Computations*, The IBM Research Symposia Series. Springer US. 85–103.

Lang, J., and Zanuttini, B. 2012. Knowledge-based programs as plans - the complexity of plan verification. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 504–509.

Nebel, B., and Bäckström, C. 1994. On the computational complexity of temporal projection, planning, and plan validation. *Artificial Intelligence* 66(1):125–160.

Penberthy, S. J., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR 1992)*. 103–114.