On the Computational Complexity of Plan Verification, (Bounded) Plan-Optimality Verification, and Bounded Plan Existence

Songtuan Lin1, Conny Olz2, Malte Helmert3, Pascal Bercher1

1 School of Computing, The Australian National University
2 Institute of Artificial Intelligence, Ulm University
3 Department of Mathematics and Computer Science, University of Basel
{songtuan.lin, pascal.bercher}@anu.edu.au, conny.olz@uni-ulm.de, malte.helmert@unibas.ch

Abstract

In this paper we study the computational complexity of several reasoning tasks centered around the bounded plan existence problem. We do this for standard classical planning and hierarchical task network (HTN) planning and each for a grounded and a lifted representation. Whereas bounded plan existence complexity is known for classical planning, it has not yet been studied for HTN planning. For plan verification, results were available for both formalisms except for the lifted HTN planning. We will present lower and upper bounds of the complexity of plan verification in lifted HTN planning and provide some new insights into its grounded counterpart, in which we show that verification is not just NP-complete in the general case, but already for a severely restricted special case. Finally, we show the complexity concerning verifying the optimality of a given plan and discuss its connection to the bounded plan existence problem.

Introduction

Automated planning is the task of finding a course of actions called a plan which achieves a certain goal. An immense effort has been devoted to studying the computational complexity of the plan existence problem in the context of both non-hierarchical (classical) planning (Erol, Nau, and Subrahmanian 1991; Bylander 1994; Helmert 2006; Bäckström and Jonsson 2011) and hierarchical planning (Erol, Hendler, and Nau 1996; Geier and Bercher 2011; Alford et al. 2014; Alford, Bercher, and Aha 2015a,b; Bercher, Lin, and Alford 2022) which is to decide whether a planning problem has a solution. In contrast, the number of research endeavors on the complexity of deciding whether there exists a plan up to a certain length (the bounded plan existence problem) is relatively small which is a standard way to frame the problem of finding an optimal plan as a decision problem. In particular, the complexity results only exist in the classical setting but not in the hierarchical one, despite that many approaches related to this problem have been developed for both formalisms (Karpas and Domshlak 2009; Pommerening et al. 2014; Bercher et al. 2017; Behnke, Höller, and Biundo 2019; Behnke and Speck 2021).

We will discuss in this paper the complexity results of several problems centered around the bounded plan existence problem. Our discussion will start with the plan verification problem, which serves as the basis for the investigation of the bounded plan existence problem, and ends up with the plan optimality verification problem and its extension, the bounded plan optimality verification problem. Plan optimality verification is to verify whether a plan is an optimal solution to a planning problem, and its bounded version is to check whether the length of a given plan is not far away from the length of an optimal one by some bound.

We will investigate some general properties of these problems and discuss their complexity results in the specific context of classical planning (Ghallab, Nau, and Traverso 2004) and Hierarchical Task Network (HTN) planning (Bercher, Alford, and Höller 2019), which is the most commonly used hierarchical planning (Ghallab, Nau, and Traverso 2004; Bercher, Alford, and Höller 2019) formalism. One important reason for discussing all these results, which are summarized in Tab. 1, is that they can serve as a reference for future research endeavors in related disciplines.

Concretely, for plan verification, although the complexity is well-developed for classical planning and grounded HTN planning (Behnke, Höller, and Biundo 2015), no investigations have been done for lifted HTN planning. Here, we will present a lower and an upper bound of the complexity of lifted HTN plan verification, which turns out to be significantly harder compared to its grounded counterpart.

For the bounded plan existence problem, we will discuss its complexity in terms of both the encoding size and the magnitude of the bound. For this, we follow the methodology by Bäckström and Jonsson (2011) which encodes the bound in binary and in unary, respectively. Lastly, we will discuss the connection between the bounded plan existence problem and the plan optimality verification problem and present the complexity results for the latter.

Background

We first present the notations which will be used throughout the paper and the planning formalisms on which the complexity results are developed.

Size of Objects

Given an arbitrary object $x$, e.g., $x$ can be a number, a problem instance, etc., we say that the size of $x$, written $\|x\|$, is the length of a binary string which encodes the object $x$. When studying the complexity of a problem,
Lifted Classical Planning is an extension of the grounded one and is defined as propositions, i.e., \( \pi \) and for each \( \langle \pi, s \rangle \), from one state \( s \) into another tentative state \( t \) in classical planning is a set of \( v \) and \( \alpha \) of \( \text{The Objective of (grounded) classical planning is a tuple} \ \Pi = (D, s_I, g) \ \text{where} \ D = (\mathcal{F}, A, \alpha) \ \text{is called the domain of} \ \Pi. \ \mathcal{F} \ \text{is a (finite) set of propositions,} \ A \ \text{is a (finite) set of action names (or actions for short), and} \ \alpha: A \rightarrow 2^F \times 2^F \times 2^F \ \text{is a function mapping each action} \ a \ \in A \ \text{to its precondition, add list, and delete list, written} \ a(a) = (\text{prec}(a), \text{add}(a), \text{del}(a)). \ s_I \ \in 2^F \ \text{is the initial state of} \ \Pi \ \text{and} \ g \subseteq \mathcal{F} \ \text{is the goal description.}

Generally speaking, the objective of (grounded) classical planning is to find an action sequence which turns the initial state into another state where the goal description is satisfied. Formally, a state \( s \) in classical planning is a set of propositions, i.e., \( s \in 2^F \). Applying an action \( a \in A \) in a state \( s \) will result in a new state \( s' \) with \( s' = (s \backslash \text{del}(a)) \cup \text{add}(a) \). An action \( a \) is applicable in a state \( s \) if \( \text{prec}(a) \subseteq s \). In other words, the precondition of \( a \) is satisfied in \( s \). For convenience, we write \( s \rightarrow_a s' \) to indicate that the action \( a \) is applicable in the state \( s \), and the state \( s' \) is obtained by applying \( a \) in \( s \). Further, given a state \( s \) and an action sequence \( \pi = (a_1 \cdots a_n) \) (\( n \in \mathbb{N} \)), we write \( s \rightarrow^*_\pi s' \) for some state \( s' \) to indicate that \( s' \) is obtained by applying \( \pi \) in \( s \), that is, there exists a state sequence \( (s_0 \cdots s_n) \) such that \( s_0 = s, s_n = s' \), and for each \( 1 \leq i \leq n, s_{i-1} \rightarrow_a s_i \). Consequently, a solution to a (grounded) classical planning problem is an action sequence \( \pi \) such that \( s_I \rightarrow^*_\pi s' \) for some state \( s' \) with \( g \subseteq s' \).

Lifted Classical Planning The lifted classical planning formalism is an extension of the grounded one and is defined on the alphabet of a first-order language \( \Sigma = (\mathcal{V}, \mathcal{O}, \mathcal{R}) \) in which \( \mathcal{V} \) is a set of variables, \( \mathcal{O} \) a set of objects, and \( \mathcal{R} \) a set of predicates. A predicate \( p \in \mathcal{R} \) is of the form \( p = P(v_1, \cdots, v_n) \) for some number \( n \) where \( P \) is called the predicate’s name, and \( v_i \in \mathcal{V} \) for each \( 1 \leq i \leq n \). Substituting every variable in a predicate with an object is called grounding the predicate. It is characterized by a variable substitution function \( g: \mathcal{V} \rightarrow \mathcal{O} \). More concretely, given a variable substitution function \( g \), grounding the predicate \( p \) according to \( g \) results in the grounded predicate, written \( p[g] \), with \( p[g] = P(g(v_1), \cdots, g(v_n)) \). In particular, a grounded predicate is equivalent to a proposition in the grounded classical planning formalism.

A lifted planning problem is again a tuple \( \Pi = (D, s_I, g) \) with \( D = (\Sigma, A, \alpha) \) being its domain. In the lifted setting, \( A \) is a set of action schemas. An action schema, \( a \in A \), also consists of an action name and a tuple of variables, written \( A(v_1, \cdots, v_n) \) (\( n \in \mathbb{N} \)) with \( A \) being the action name. \( \alpha \) maps an action schema to its precondition, add list, and delete list, written \( \alpha(a) = (\text{prec}(a), \text{add}(a), \text{del}(a)) \). Each of which is a set of predicates \( P(v_1, \cdots, v_i) \) such that \( v_i \in \{v_1, \cdots, v_n\} \) for each \( r \in \{1, \cdots, j\} \).

An action schema \( a \) can also be grounded into an action \( a \) in the grounded setting by a variable substitution function \( g \), written \( a = a[g] \). When grounding an action schema, all predicates in its precondition and add and delete list are grounded simultaneously by the same substitution function.

Lastly, \( s_I \) and \( g \) are two sets of grounded predicates (i.e., propositions) which are the initial state and the goal description of \( \Pi \), respectively. A solution to \( \Pi \) is an action sequence \( \pi = (a_1 \cdots a_n) \) such that \( s_I \rightarrow^*_\pi s' \) for some state \( s' \) with \( g \subseteq s' \), and for each \( a_i \) with \( 1 \leq i \leq n \), there exist an action schema \( a \in A \) and a variable substitution function \( g \) such that \( a_i = a[g] \).

In some literatures, the set \( \mathcal{O} \) of objects is not defined as

<table>
<thead>
<tr>
<th>Classical</th>
<th>Lifted</th>
<th>Hierarchical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grounded</td>
<td>Lifted</td>
<td>Grounded</td>
</tr>
<tr>
<td>In ( P )</td>
<td>In ( P )</td>
<td>( k ) in binary</td>
</tr>
<tr>
<td>NEXPTIME</td>
<td>NEXPTIME</td>
<td>NEXPTIME</td>
</tr>
<tr>
<td>Prop. 2</td>
<td>Prop. 4</td>
<td>Prop. 5</td>
</tr>
<tr>
<td>PSPACE</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
<tr>
<td>Prop. 3</td>
<td>Prop. 4</td>
<td>Prop. 5</td>
</tr>
</tbody>
</table>

Table 1: Summary of the complexity results and the respective theorems. All complexity results are complete except for those cases where hardness and membership are explicitly specified. Note that we demand here that a solution to an HTN planning problem is an action sequence, which is different from the standard solution criteria where a solution is a primitive task network. The complexity of the \( k \)-length plan existence with \( k \) encoded in binary in grounded classical planning was proved by Bylander (1994) while the result for the lifted setting was proved by Erol, Nau, and Subrahmanian (1991). When \( k \) is encoded in unary, the complexity in grounded classical planning was proved by Bäckström and Jonsson (2011). The complexity of plan verification in grounded HTN planning was proved by Behnke, Höller, and Biundo (2015). We show that it holds even in a restricted case.
a component of a domain but in conjunction with an initial state and a goal description, forming a planning task. This allows a domain to be paired with different planning tasks, constituting different planning problems. We adapt a more simple formalism because our complexity investigation only focuses on one single problem instance.

Notably, one can obtain a grounded planning problem \( \Pi \) from a lifted one \( \Pi' \) by grounding every predicate and action schema with all possible variable substitution functions, and the problem \( \Pi \) produced in such a way has the same solution set as \( \Pi' \). One important remark is that \( \| \Pi \| \) is exponential in \( \| \Pi' \| \), that is, \( \| \Pi \| = O(2^{|\Pi'|^q}) \) for some constant \( q \in \mathbb{N} \).

**Grounded HTN Planning** We now reproduce the formalism of the grounded Hierarchical Task Network (HTN) planning (Bercher, Alford, and Höller 2019). A grounded HTN planning problem \( \Pi \) is a tuple \( (D, s_1, tn_1, g) \) with \( D = (F, A, C, M, \alpha) \) being its domain. A grounded HTN planning problem is an extension of a grounded classical one in the sense that \( F, A, \alpha, s_1, \) and \( g \) are defined in the same way as their counterparts in the classical setting. An action \( a \in A \) in HTN planning is also called a primitive task. Two components, \( C \) and \( M \), which are not in the classical formalism, are the set of compound tasks and of methods, respectively. A method \((c, tn) \in M\) decomposes a compound task \( c \in C \) into a so-called task network \( tn \), which is essentially a partial order \( \Pi \) of primitive and compound tasks. Formally, a task network \( tn \) is a triple \( (T, \prec, \gamma) \) where \( T \) is a set of identifiers, \( \prec \subseteq T \times T \) is a partial order defined over \( T \), and \( \gamma : T \rightarrow A \cup C \) is a function that maps each identifier to a task. Two task networks, \( tn = (T, \prec, \gamma) \) and \( tn' = (T', \prec', \gamma') \), are said to be isomorphic, written \( tn \cong tn' \), if there exists a bijective mapping \( \varphi : T \rightarrow T' \) such that \( \gamma(t) = \gamma'(\varphi(t)) \) for any \( t \in T \), and for any \( t, t' \in T \), \( t \prec t' \) if and only if \( \varphi(t) \prec' \varphi(t') \). The last component \( tn_1 \) is the initial task network.

The notion of decomposing a compound task can also be extended to decomposing a task network. A task network \( tn \) with \( tn = (T, \prec, \gamma) \) is decomposed into another one \( tn' = (T', \prec', \gamma') \) by some method \( m = (c, tn) \), written \( tn \Rightarrow_m tn' \), if there exists an identifier \( t \in T \) and a task network \( tn^* = (T^*, \prec^*, \gamma^*) \) with \( tn^* \cong m \) such that

1) \( T^* \cap T = \emptyset \)
2) \( \gamma(t) = c \)
3) \( \prec = (\prec \setminus \prec_t) \cup \prec^* \cup \prec_{\delta} \) in which \( \prec_t \) is the set of ordering constraints \( \{ (t', t) | (t', t) \in \prec \} \cup \{ (t', t') | (t, t') \in \prec \} \), i.e., \( \prec_t \) is the set of all ordering constraints in \( tn \) that are associated with \( t \), and \( \prec_{\delta} \) specifies the position of \( tn^* \) in \( tn \) with respect to the task \( t \) replaced by it. In other words, \( \prec_{\delta} \) is the set \( \{ (t_1, t_2) \mid t_2 \in T^*, (t_1, t) \in \prec \} \cup \{ (t_2, t_1) \mid t_2 \in T^*, (t_1, t) \in \prec \} \).

Further, let \( tn \) and \( tn' \) be two task networks and \( m \) a sequence of methods. We use \( tn \Rightarrow_m tn' \) to indicate that \( tn' \) is obtained from \( tn \) by applying \( m \).

Like classical planning, (grounded) HTN planning is also to find an action sequence (i.e., a plan) which turns \( s_1 \) into a state satisfying \( g \). However, in HTN planning, such a plan must be obtained from the initial task network by decompositions. Concretely, a plan \( \pi \) is a solution to an HTN planning problem \( \Pi \) if \( s_1 \Rightarrow_{\pi}^* s \) with \( g \subseteq s \) for some state \( s \), and there exists a task network \( tn = (T, \prec, \gamma) \) such that \( tn_1 \Rightarrow_{\pi}^* tn \) for some method sequence \( \bar{m} \), and \( tn \) has a linearization \( \bar{tn} \) that forms \( \pi \). A linearization \( \bar{tn} = (t_1 \cdots t_{|T|}) \) of \( tn \) is a total order of \( T \) which respects \( \prec \), and by \( \bar{tn} \) forming \( \pi \), we mean that \( \pi = (\gamma(t_1) \cdots \gamma(t_{|T|})) \). For convenience, we use \( \gamma(t_{|T|}) \) to denote the task sequence formed by \( \bar{tn} \). Please note that there is a minor difference compared to standard HTN literature (Bercher, Alford, and Höller 2019; Erol, Hendler, and Nau 1996) in our solution definition. In our definition, a solution is an action sequence, which we argue makes the most sense. In standard literature, a solution is a primitive task network having an executable linearization.

**Lifted HTN Planning** A lifted HTN planning problem is a tuple \( \Pi = (D, s_1, tn_1, g) \) with \( D = (\Sigma, A, C, \alpha) \) being its domain where \( \Sigma = (V, O, R) \), \( A \), and \( \alpha \) are defined in the same way as that in lifted classical planning. Every action schema is also called a primitive task schema. \( C \) is now a set of compound task schemas and \( \alpha \) a set of method schemas. A compound task schema \( c \in C \) is simply a compound task name together with a tuple of variables. A method schema \( m \) is a tuple \((c, tn)\) where \( c \) is a compound task schema and \( tn \) a task network schema. A task network schema is again a tuple \((T, \prec, \gamma)\) where \( T \) and \( \prec \) are identical to those in a grounded task network, and \( \gamma \) maps each identifier to a task schema.

A task, task network, or method schema \( x \) can again be grounded by some variable substitution function \( \varphi : \emptyset \rightarrow O \), written \( x[\varphi] \). When grounding a task network schema \( tn \) with a substitution function \( \varphi \), all task schemas in \( tn \) are grounded simultaneously by \( \varphi \), and for any method schema \( m = (c, tn) \) with \( m[\varphi] = \langle c[\varphi], tn[\varphi] \rangle \). A grounded task schema and a grounded method schema are equivalent to a task and a method in the grounded setting, respectively.

\( s_1 \) and \( g \) are again the initial state and the goal description consisting of propositions, and \( tn_1 \) is the grounded initial task network. An action sequence \( \pi \) is a solution to a lifted HTN planning problem if \( s_1 \Rightarrow_{\pi}^* s \) for some state \( s \) with \( g \subseteq s \), and there exists a grounded method sequence \( \bar{m} = \langle m_1 \cdots m_n \rangle, n \in \mathbb{N} \) such that for each \( 1 \leq i \leq n \), there exists a method schema \( m_i \in \mathcal{M} \) with \( m_i[\varphi] = m_i \) for some \( \varphi \), and \( tn_1 \Rightarrow_{\varphi}^* tn \) for some primitive grounded task network \( tn \) which possesses a linearization forming \( \pi \).

Similar to lifted classical planning, one could also ground a lifted HTN planning problem without changing its solution set, and the size of the grounded problem is again exponential in that of the lifted one.

**Proposition 1.** Let \( \Pi \) be a lifted (classical or hierarchical) planning problem and \( \Pi \) its grounded counterpart. Then it holds that \( \| \Pi \| = O(2^{|\Pi'|^q}) \) for some constant \( q \in \mathbb{N} \).

**Plan Verification**

We move on to discuss the complexity for plan verification, which is to decide, given a planning problem and a plan, whether the plan is a solution to the planning problem.

The complexity results for classical planning are obvious. In the grounded setting, a plan can clearly be validated.
in polynomial time by checking whether it is executable and satisfies all goals. This is well-known and exploited by verifiers like VAL (Howey, Long, and Fox 2004). Given a ground plan but a lifted problem description, the problem gets slightly more complicated because for each action in the plan we need to check whether it can be created by grounding some lifted action schema. This can easily be checked in polynomial time (just match constants to the respective variables). As a result, the plan verification problem for both grounded and lifted classical planning is in \( \mathbb{P} \).

In contrast, the plan verification problem in HTN planning is more computationally expensive. Previous works have already shown that it is already \( \mathbb{NP} \)-complete in the grounded setting (Behnke, Höller, and Biundo 2015; Bercher, Lin, and Alford 2022). Those investigations rely on the standard definition of solutions being task networks that possess some executable linearization, whereas we define such a linearization as the solution itself. However, even with our definition of solutions, grounded HTN plan verification is still \( \mathbb{NP} \)-complete (Behnke, Höller, and Biundo 2015, Thm. 2).

The existing hardness proof (Behnke, Höller, and Biundo 2015) for plan verification (with our definition of solutions) relies on finding a decomposition hierarchy that results in the given plan. We can further improve this result by showing that \( \mathbb{NP} \)-hardness holds even if the initial task network is primitive. This follows from the fact that deciding whether a sequence of tasks is a linearization of a partial order task network is \( \mathbb{NP} \)-complete (Lin and Bercher 2023). The result by Lin, Grastien, and Bercher (2023) does not differentiate primitive and compound tasks, which makes the grounded HTN plan verification problem with a primitive initial task network a special case of that problem.

**Proposition 2.** The plan verification problem for grounded HTN planning is \( \mathbb{NP} \)-complete. This holds even in the special case where the initial task network of the given planning problem is primitive.

As a special case, the plan verification problem in the context of total order (TO) HTN planning is poly-time decidable (Behnke, Höller, and Biundo 2015). A TOHTN planning problem is such that the initial task network is totally ordered, and every method refines a compound task into a total order task network as well. Solving a TOHTN planning problem is computationally cheaper than solving a partial order one. Many theoretical investigations into properties of TOHTN planning have been made which have great potential to be utilized to solve TO problems more efficiently (e.g., see the work done by Olz, Biundo, and Bercher (2021) and by Olz and Bercher (2023)). The poly-time decidability of the TOHTN plan verification problem holds because a (grounded) TOHTN planning problem is essentially equivalent to a context-free grammar (CFG) (Höller et al. 2014), and hence, the plan verification problem is equivalent to the parsing problem for a CFG. Bearing this connection, several efficient TOHTN plan verifiers (Barták et al. 2021; Lin et al. 2023; Pantucková and Barták 2023) have been developed by exploiting CFG parsers.

One might raise the question asking why grounded HTN plan verification is \( \mathbb{NP} \)-complete but not \( \mathbb{PSPACE} \)-complete because it is \( \mathbb{PSPACE} \)-complete to decide whether a word is in a context-sensitive language (CSL) (Immerman 1988; Szepesváry 1987) while the solution set of a grounded HTN problem is a CSL (Höller et al. 2014). It is because the language, denoted \( L_1 \), of the solution set of a grounded HTN problem \( \mathcal{P} \) is different from the language of the word membership problem for CSLs, denoted \( L_2 \). More specifically, \( L_1 \) is \( \{ \pi \mid \pi \in \text{sol}(\mathcal{P}) \} \) whereas \( L_2 \) is the set \( \{ (\omega, \Gamma) \mid \omega \in \Gamma \} \) in which \( \Gamma \) is a context-sensitive grammar and \( \omega \) a word. That is, every element in \( L_2 \) is a tuple of a word and a grammar, but every element in \( L_1 \) is only a plan. \( L_2 \) is \( \mathbb{PSPACE} \)-complete while the complexity of \( L_1 \) depends on \( \mathcal{P} \).

Next, we extend our investigation from the grounded setting to the lifted one. Note that in the lifted setting, the plan to be verified is still grounded, but the planning problem is represented in the lifted way. Unlike the case in classical planning, hardness of the plan verification problem increases dramatically in lifted HTN planning.

**Theorem 1.** The plan verification problem in the context of lifted HTN planning is \( \mathbb{PSPACE} \)-hard.

**Proof.** We will reduce from the grounded classical plan existence problem. Let \( \Pi = (D, s, f, g) \) with \( D = (F, A, \omega) \) be a grounded classical planning problem. For convenience, we assume that \( F = \{ p_1, \ldots, p_n \} \) with \( n \in \mathbb{N} \). Our objective is thus to construct a lifted HTN planning problem \( \Pi \) and a plan \( \pi \) such that \( \pi \) is a solution to \( \Pi \) if and only if \( \Pi \) has a solution. To this end, we need to construct the following components (gadgets) that can 1) simulate states in the classical problem, 2) encode actions in \( A \), and 3) encapsulate the initial state and the goal of the classical problem.

For the purpose of simulating states in the classical problem, we construct one compound task schema \( c \) such that each grounding of this schema represents a state. This compound task schema \( c \) is constructed as follows:

\[
c = \text{State}(x_1, \ldots, x_n, v_0, v_1)
\]

where \( x_1, \ldots, x_n, v_0, v_1 \) are variables, and \( n = |F| \). Each \( x_i \) (\( 1 \leq i \leq n \)) represent the respective proposition \( p_i \in F \).

Our constructed lifted HTN problem contains only two objects, namely, 0 and 1, and hence, for any grounded version of \( c \), if a variable \( x_i \) is grounded to 1, then it means that the proposition \( p_i \) is in the respective state. For the remaining two variables \( v_0 \) and \( v_1 \), their purpose is to simulate actions’ executions in the classical problem, and our latter construction will ensure that \( v_0 \) can only be grounded to 0 and \( v_1 \) to 1 (we will discuss this in more detail shortly).

Next we discuss how to encode the actions in the classical problem. This is done by constructing method schemas. For each action \( a \in A \), we construct a method schema \( m_a \), decomposing the task schema \( \text{State}(x'_1, \ldots, x'_n, v_0, v_1) \) into a task network schema which contains solely one compound task schema \( \text{State}(x'_1, \ldots, x'_n, v_0, v_1) \) such that for all \( 1 \leq i \leq n \), 1) \( x'_i = v_1 \) if \( p_i \in \text{pre}(a) \), 2) \( x'_i = v_0 \) if \( p_i \in \text{del}(a) \), 3) \( x'_i = v_1 \) if \( p_i \in \text{add}(a) \), and 4) \( x'_i = x'_i \) if none of the previous holds. As mentioned above, we will enforce that \( v_0 = 0 \) and \( v_1 = 1 \). Thus, all variables \( x'_i \) with \( x'_i = v_1 \) together restrict that in order to use this method...
schema to decompose a grounded task State, the respective \( x_i \) in the task (to be decomposed) must be grounded to 1. This thus simulate the constraint in the classical problem that an action \( a \) is applicable in a state \( s \) if \( \text{prec}(a) \subseteq s \). Similarly, those \( x_i^j \)'s with \( x_i^j = v_1 \) (resp. \( x_i^j = v_0 \)) encode that the respective propositions will be added to (resp. deleted from) the state \( s \) after applying the action \( a \).

Lastly, we present the encoding for the classical problem’s initial state \( s_I \) and goal \( g \). \( s_I \) is encoded by the initial task \( c_I \) of the lifted HTN problem, which is constructed as follows:

\[
c_I = \text{State}(y_1, \ldots, y_n, 0, 1)
\]

where for each \( 1 \leq i \leq n \), \( y_i = 1 \) if the respective \( p_i \in s_I \), otherwise, \( y_i = 0 \). In particular, since we let \( v_0 = 0 \) and \( v_1 = 1 \) in \( c_I \), we enforce that the values of these two variables cannot be changed in decomposition because in each method schema we construct, \( v_0 \) and \( v_1 \) are always inherited down from the task schema to be decomposed to the subtask.

The goal in the classical planning problem is represented by the plan \( \pi \) to be verified. To construct this plan, we first need to construct \( n \) extra compound task schemas \( P_i(x) \)'s, one for each \( p_i \in \mathcal{F} \), and one method schema \( m_s \) which decomposes the task schema \( \text{State}(x_1, \ldots, x_m, v_0, v_1) \) into the total order task network \( \langle P_1(x_1) \cdots P_n(x_n) \rangle \). The purpose of these \( P_i \)’s and \( m_s \) is to extract each proposition \( p_i \) from the respective state. Each \( P_i(x) \) can further be decomposed by two method schemas. One decomposes \( P_i(x) \) into the task network consisting of only one primitive task schema \( \text{Exist}P_i(x) \) that has neither precondition nor effects. The other method schema decomposes \( P_i(x) \) into the primitive task schema \( \text{NotCare}(x) \), which again has no precondition and effects. Each \( \text{Exist}P_i(x) \) can be viewed as a certificate of whether the proposition \( p_i \) is in the respective state, namely, \( \text{Exist}P_i(1) \) asserts the presence of \( p_i \), and \( \text{Exist}P_i(0) \) asserts its absence. \( \text{NotCare}(x) \) simply means that we don’t care whether the proposition is in the state. Finally, the plan \( \pi \) to be verified is the sequence \( \langle a_1 \cdots a_n \rangle \) where for each \( 1 \leq i \leq n \), \( a_i = \text{Exist}P_i(1) \) if \( p_i \in g \), otherwise, \( a_i = \text{NotCare}(x) \). The interpretation of the plan \( \pi \) is that for every proposition in the goal of the classical problem, we must assert its presence in the final state obtained by executing a solution plan, and for those propositions that are not in the goal, we do not care whether they hold or not.

By construction, there exists a plan \( \pi' \) in the \( \Pi \) such that \( s_I \rightarrow_{\pi'} s \) for some \( s \) with \( g \subseteq s \) iff there exists a decomposition hierarchy in the constructed lifted HTN problem that can decompose \( c_I \) into the constructed plan \( \pi \).

As an example of the reduction, consider a grounded classical problem which has three propositions \( \{p_1, p_2, p_3\} \) and three actions \( \{a_1, a_2, a_3\} \). The initial state \( s_I \) of the classical problem is \( \{p_1\} \), and the goal \( g \) is \( \{p_2\} \). The precondition and effects of each action are depicted in Fig. 1, which also depicts the construction of each method schema that encodes the respective action. Fig. 2 depicts how decomposition simulates actions’ executions. Concretely, the initial task \( c_{a_1} \) encoding \( s_I \) is \( \text{State}(1, 0, 0, 1) \) because \( s_I \) has only \( p_1 \). \( a_1 \) is the only action which is applicable in \( s_I \). Thus, only \( m_{a_1} \) has a grounded version that can decompose \( c_I \). For the action \( a_2 \), since it requires \( p_3 \) which is not in \( s_I \), it leads to a contradiction that \( v_1 \) should be grounded to both 0 and 1 simultaneously. The similar situation also happens to \( a_3 \). Decomposing \( c_I \) leads to the task \( \text{State}(0, 0, 1, 0, 1) \) which encodes the state obtained by applying \( a_1 \) in the initial state.

The plan \( \pi \) to be verified that encodes the goal of the classical problem is \( \langle t_1 t_2 t_3 \rangle \) where \( t_2 = \text{Exist}P_2(1) \) and \( t_3 = \text{NotCare}(x) \). This plan is a solution to the lifted HTN problem. The decomposition resulting in \( \pi \) is to apply in sequence \( m_s, m_{a_1}, m_{a_2}, m_{a_3} \), each with the corresponding grounding, to obtain the task \( \text{State}(0, 1, 1, 0, 1) \), which can be further decomposed into \( \pi \). Note that this decomposition also simulates the solution to the classical problem. However, if the goal of the classical problem is \( \{p_1, p_2\} \) (in which case the classical problem is unsolvable), the plan \( \pi \) shall be \( \langle \text{Exist}P_1(1) \text{Exist}P_2(1) \text{NotCare}(x) \rangle \). Now this plan is not a solution to the HTN problem because no decomposition can ground \( x_1 \) in \( \text{State} \) to 1.

For membership, one can observe that the lifted HTN planning veriﬁcation problem is in \textbf{NEXPTIME}. This is because for any lifted HTN planning problem \( \Pi \) and a plan \( \pi \), we can ﬁrst ground \( \Pi \) into a grounded one \( \Pi' \) in exponential time according to Prop. 1. Since the grounded HTN planning problem is in \textbf{NP}, we can non-deterministically verify whether \( \pi \) is a solution to \( \Pi \) in polynomial time with respect to \( |\Pi'| \) and \( ||\pi|| \). It thus follows that whether \( \pi \) is a solution to \( \Pi \) can be checked non-deterministically in exponential time with respect to \( |\Pi'| \).

**Theorem 2.** The plan veriﬁcation problem in lifted HTN planning is \textbf{PSPACE}-hard and is in \textbf{NEXPTIME}.

**Bounded Plan Existence**

We now move on to discuss the complexity of the bounded \((k\text{-length})\) plan existence problem, which is to decide, given
a planning problem and a \( k \in \mathbb{N} \), whether there is a solution plan \( \pi \) to the problem of length up to \( k \). We start with some general properties of this problem and then discuss its complexity in specific planning formalisms.

One insight into this problem is that it can always be decided by a two-step procedure independent of any planning formalism: We first guess a plan \( \pi \) of length up to the bound \( k \) and then verify whether \( \pi \) is a solution to the given planning problem \( \Pi \). Bearing this observation, one could notice that the complexity of the bounded plan existence problem is restricted by the formula:

\[
T_N^\Pi (|\Pi| + |\pi|) + T_G^\pi (|k|)
\]

where \( T_N^\Pi \) is a function that denotes the runtime with respect to the encoding size of \( \Pi \) and \( \pi \) for verifying whether \( \pi \) is a solution, and \( T_G^\pi \) denotes the runtime with respect to the encoding size of \( k \) for guessing \( \pi \). The superscript \( N \) indicates that both operations are done non-deterministically.

Having Formula (1) in hand, one could observe that \textsc{NEXPTIME}-membership of the bounded plan existence problem holds for any planning formalism that has the following two properties: 1) each action can be encoded in polynomially many bits with respect to the size of a planning problem, which ensures that \( T_G^\pi (|k|) \) is exponential to \( |k| \), and 2) verifying whether a plan is a solution to a planning problem is in \textsc{NP} with respect to the encoding size of the plan and the planning problem, which further ensures that \( T_N^\Pi (|\Pi| + |\pi|) \) is also exponential to \( |k| \).

One could observe that classical planning (including both the grounded and the lifted representation) satisfies these two properties, which implies \textsc{NEXPTIME}-membership. In fact, the tight bound for the grounded representation have been proved earlier (Erol, Nau, and Subrahmanian 1991; Bylander 1994), which is \textsc{PSPACE}-complete, making it as hard as its unbounded version (Bylander 1994). \textsc{NEXPTIME}-completeness for the lifted setting has also been proved by Erol, Nau, and Subrahmanian (1991) whereas its unbounded counterpart is \textsc{EXPSPACE}-complete. We can extend this reasoning to grounded HTN planning.

**Theorem 3.** The \( k \)-length (bounded) plan existence problem for grounded HTN planning is \textsc{NEXPTIME}-complete.

**Proof.** Membership can be obtained by recognizing the fact that grounded HTN planning satisfies the two properties that assert \textsc{NEXPTIME}-membership. For hardness, we reduce from the grounded acyclic HTN plan existence problem. The basis for the reduction is the result by Behne et al. (2016) that for any acyclic HTN planning problem, the length of a solution is bounded by an exponential number \( k^* \). Thus, by letting \( k = k^* \), deciding whether an acyclic HTN problem has a solution is equivalent to deciding whether that problem has a solution bounded in length by \( k \).

For lifted HTN planning, Thm. 2 implies that \( T_N^\Pi (|\Pi| + |\pi|) \) is exponential to \( |\pi| \) and hence is double-exponential to \( |k| \). We thus have the following result.

**Corollary 1.** The \( k \)-length plan existence problem for lifted HTN planning is \textsc{NEXPTIME}-hard and in 2\textsc{NEXPTIME}.

Note that hardness follows from the fact that a grounded HTN problem can be viewed as a special case of a lifted one.

**Encoding the Bound in Unary** Our discussion about the \( k \)-length plan existence problem so far is restricted to the case where the bound \( k \) is given in binary. That is, the magnitude of \( k \) grows exponentially in its encoding size. This however might contradict the intention of giving such a bound. More concretely, in practice, when a user uses a planner to find a plan of length up to a certain bound, the user is actually concerned with the magnitude of this bound but not the encoding size. For instance, the input \( k \) in practice would be increased incrementally, that is, we would increase its magnitude by one or two (etc.) but not exponentially.

Bearing this scenario, Bäckström and Jonsson (2011) investigated the \( k \)-length plan existence problem from a dif-
The given bound evaluates selecting actions in the classical problem. The initial methods thus also the initial task). For each problem has the same proposition set and action set as the problem that simulates it is constructed as follows: the HTN structural a)
cept that 1) for each lem \( \mathcal{\Pi} = (\{\mathcal{\Pi}\} + \|\pi\|) \) is also a polynomial. Thus, NP-membership for lifted classical planning follows immediately. Furthermore, NP-hardness in lifted classical planning holds as well due to NP-hardness in the grounded setting.

**Theorem 4.** The \( k \)-length plan existence problem for lifted classical planning is NP-complete if \( k \) is encoded in unary.

Next we move on to HTN planning.

**Theorem 5.** The grounded \( k \)-length plan existence problem for HTN planning is NP-complete, if \( k \) is encoded in unary.

**Proof.** Membership can be obtained by recognizing that the term \( T^N_{\mathcal{\Pi}}(\|\mathcal{\Pi}\| + \|\pi\|) \) in Formula (1) is a polynomial with respect to the encoding size of \( k \) due to its unary encoding.

For hardness, we can reduce from the grounded classical \( k \)-length plan existence problem with \( k \) given in unary. The reduction is done by using the construction by Erol, Hendler, and Nau (1996) that simulates a grounded classical problem with a grounded HTN one. Given a grounded classical problem \( \mathcal{\Pi} = (\mathcal{D}, s_1, g) \) with \( \mathcal{D} = (\mathcal{F}, \mathcal{A}, \alpha) \), the grounded HTN problem that simulates it is constructed as follows: the HTN problem has the same proposition set and action set as the classical problem and only one compound task \( c \) (which is thus also the initial task). For each \( a \in \mathcal{A} \), we construct two methods \( m_1 \) and \( m_2 \) where \( m_1 \) decomposes \( c \) into a task network having only one action \( a \), and \( m_2 \) decomposes \( c \) into the total order task network \( (c, c) \). Such a construction simulates selecting actions in the classical problem. The initial state and the goal of the HTN problem are also identical to the classical one. The reduction can then be done by copying the given bound \( k \) (in unary).

**Theorem 6.** The \( k \)-length plan existence problem in lifted HTN planning with \( k \) given in unary is PSPACE-hard and is in \textit{NEXPTIME}.

**Proof.** Thm. 2 implies that the term \( T^N_{\mathcal{\Pi}}(\|\mathcal{\Pi}\| + \|\pi\|) \) in Formula (1) is exponential to \( |k| \) due to the unary encoding. As a result, \textit{NEXPTIME}-membership holds. For hardness, we again reduce from the grounded classical plan existence problem. The construction of the lifted HTN problem is identical to the one presented in the proof for Thm. 1 except that 1) for each \( p_i \in \mathcal{F} \) in the classical problem, we construct a lifted predicate \( \textit{Pred}_i(x) \), 2) for each proposition \( p \in \mathcal{F} \), the respective primitive task schema \( \textit{Exist}\Phi_i(x) \) now has a single positive effect \( \textit{Pred}_i(x) \), and 3) the goal of the lifted HTN problem is \( \{\textit{Pred}_i(1) \mid 1 \leq i \leq n, p_i \in \mathcal{G}\} \).

Every proposition \( \textit{Pred}_i(1) \) can only be obtained from the respective primitive task \( \textit{Exist}\Phi_i(1) \). Hence, the lifted HTN problem has a solution iff there exists a decomposition hierarchy that results in a plan containing all \( \textit{Exist}\Phi_i(1) \)’s with \( p_i \in \mathcal{G} \), which can happen iff there exists an action sequence in the classical problem that can turn the initial state into another state where the goal is satisfied, by our argument in the proof for Thm. 1 (in this supplementary material). Furthermore, notice that any plan produced by the lifted HTN problem always has length \( n \). Hence, we could let the bound \( k \) be \( n \), which thus complete the reduction.

**Verification of Plan Optimality**

Lastly, we turn to discuss the problem of plan optimality verification, which is to decide, given a planning problem and a plan, whether there exist no other solutions of length smaller than that of the given one. Many vital tasks are centered on plan optimality verification, e.g., the task of model reconciliation, of plan post-optimization, and of domain learning. The first one is to change a planning problem’s domain with the least number of changes so as to turn a plan into an optimal solution, which is \textit{NP}-complete (Bachor and Behnke 2024). The second one is concerned with whether a plan can be further optimized by removing some redundant actions from it, which is \textit{NP}-complete in both classical planning (Fink and Yang 1992; Nakhost and Müller 2010) and POCL planning (Olz and Bercher 2019). The last task is about learning (i.e., constructing) a domain from fully observed traces (actions and states) such that in the learned domain the given plans contain no redundant actions. Depending on the notion of redundancy and additional imposed constraints, this problem can be in \textit{P}, \textit{NP}-complete, or \textit{coNP}-complete (Bachor and Behnke 2024).

Despite that the complexity results for those related problems are well-developed, the problem of plan optimality verification itself has not yet received particular attention. One remark of great importance is that the plan optimality verification problem can be viewed as a complement of the bounded plan existence problem with the bound given in unary. The reason is that each action in the plan \( \pi \) provided in the plan optimality verification problem does not matter. What we are really concerned with is the length \( |\pi| \) of that plan. Thus, asking whether the plan \( \pi \) is an optimal one is identical to asking whether there exist no solution plans of length smaller or equal to \( |\pi| + 1 \) encoded in unary, which is a complement of the bounded plan existence problem with the bound given in unary.

As a result, the complexity of the plan optimality verification problem for a specific planning formalism is naturally the complement of that of the bounded plan existence problem with the bound given in unary for that formalism.

**Proposition 3.** The plan optimality verification problem for classical planning, including both the grounded and lifted settings, and grounded HTN planning is \textit{coNP}-complete. For lifted HTN planning, this problem is \textit{PSPACE}-hard and in \textit{coNEXPTIME}. 

20209
PSPACE-hardness in lifted HTN planning holds because of the fact that PSPACE = coPSPACE (Szelepcsényi 1987; Immerman 1988).

Since optimality is often diametral to efficiency, and finding a strict optimal solution is time-consuming in practice, it is quite often the case that a solution whose length lies in an acceptable range of the length of an optimal solution is practically more desirable.

Bearing this scenario, we thus formulate the problem of bounded optimality verification, which is to decide, given a planning problem \( \Pi \), a solution plan \( \pi \) to \( \Pi \), and a bound \( k \), whether the length \( |\pi^*| \) of an optimal solution \( \pi^* \) to \( \Pi \) satisfies \( |\pi| < |\pi^*| + k \). In other words, we want to verify whether the length of \( \pi \) is not larger than the length of an optimal solution by the bound \( k \). (Note that both \( |\pi^*| \) and \( \pi^* \) are not given as input.)

Although the bounded optimality verification problem describes a scenario that is different from the one described by the plan optimality verification problem, these two problems are actually equivalent from the theoretical point of view. This is because the bounded optimality verification problem is identical to asking whether there exist no solution plans \( \pi' \) to \( \Pi \) such that \( |\pi| - k > |\pi'| \). For if such a \( \pi' \) exists, we have \( |\pi^*| \leq |\pi'| \) because \( \pi^* \) is an optimal solution, and hence, \( |\pi| > |\pi'| + k \geq |\pi^*| + k \), which is a contradiction. Consequently, for any planning formalism, the bounded optimality verification problem with \( \pi \) and \( k \) being the given plan and bound, respectively, is again the complement of the bounded plan existence problem in which the bound is \( |\pi| - k \) and is encoded in unary.

**Proposition 4.** The bounded plan optimality verification problem (with the bound given in binary) has the same complexity as the plan optimality verification problem, independent of planning formalisms.

We have already mentioned earlier that in the (bounded) plan optimality verification problem, what really matters is the length of the given plan. As a consequence, we can further generalize those problems by replacing the given plan with the length of the plan. That is, given a planning problem \( \Pi \), and two numbers \( k_a \) and \( k \) where \( k_a \) is the length of some solution, we want to decide whether there exist no solution plans \( \pi' \) to \( \Pi \) of length \( k' \) such that \( k_a - k' > k \). We argue that this generalized version is useful in the scenario of modeling assistance where a (planning) domain modeler would like to know whether a domain is correctly modeled (McCluskey, Vaquero, and Vallati 2017; Lin and Bercher 2021, 2023; Lin, Grastien, and Bercher 2023). One way to do so is by validating whether certain properties hold in the domain. In our case, one could ask whether there exists an optimal solution within a range of \( k \), provided a claim that there is a solution \( \pi \) with \( |\pi| \) steps (in some domains, the modeler might be aware that the solution \( \pi \) exists, but doesn’t want to write it down for the purpose of asking this question).

For this generalized problem, since we replace the given plan with a number, its complexity is thus the complement of the bounded plan existence problem without encoding the bound in unary, independent of planning formalisms.

**Proposition 5.** The complexity of the bounded plan optimality verification problem (with the bound given in binary) where the plan is not explicitly given, is the complement of the bounded plan existence problem, independent of planning formalisms.

When the bound is zero, the bounded plan optimality verification problem boils down to the plan optimality verification problem where a plan is replaced by its length.

**Proposition 6.** The complexity of plan optimality verification where only the plan length is given is the complement of the bounded plan existence problem (with the bound given in binary).

**Discussion**

Our complexity investigations for HTN planning are based on the solution criteria that differ from the standard ones. In fact, the presented complexity results for HTN planning remain the same even with the standard solution criteria where we demand that a solution to an HTN problem is a primitive task network that possesses an executable linearization. The reason for this is that an action sequence is a totally ordered primitive task network and hence is a special case of a partially ordered primitive task network. Thus, hardness of all problems which are investigated in this paper still hold in the context of HTN planning with the standard solution criteria. Membership of these problems does not change as well because for any partially order primitive task network, we can guess a linearization of it and verify whether this linearization is executable, which can be done in polynomial time with respect to the encoding size of the task network.

**Conclusion**

We studied the computational complexity of several questions centered at the bounded plan existence problem. Our results show that in classical planning and grounded HTN planning, the computational complexity of plan verification lies in the range of P to NP-complete, whereas it increases dramatically in lifted HTN planning. For bounded plan existence, its complexity ranges from PSPACE-complete to \( \text{coNEXPTIME} \) depending on planning formalisms whereas it decreases to the range from \( \text{NP} \)-complete to \( \text{NEXPTIME} \) when the bound is encoded in unary. For the problem of (bounded) plan optimality verification, if the plan to be verified is explicitly given, then it is the complement of bounded plan existence with the bound given in unary. If only the plan length is given, it is the complement of the bounded plan existence problem with the bound given in binary.

**Acknowledgements**

This work was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215.

**References**


