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

Songtuan Lin[1], **Conny Olz**[2], **Malte Helmert**[3], **Pascal Bercher**[1]

[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,

		Plan Verification	k-length Plan Existence		Plan Opt. Verification	Bounded Plan Opt. Verification	
			k in binary	k in unary		plan given	plan length given
Classical	ground	In **P**	**PSPACE**	**NP**	**coNP** Prop. 3	**coNP** Prop. 4	**PSPACE** Prop. 5
	lifted	In **P**	**NEXPTIME**	**NP** Thm. 4	**coNP** Prop. 3	**coNP** Prop. 4	**coNEXPTIME** Prop. 5
Hierarchical	ground	**NP** Prop. 2	**NEXPTIME** Thm. 3	**NP** Thm. 5	**coNP** Prop. 3	**coNP** Prop. 4	**coNEXPTIME** Prop. 5
	lifted	**PSPACE**-hard Thm. 1 In **NEXPTIME** Thm. 2	**NEXPTIME**-hard Cor. 1 In 2**NEXPTIME** Cor. 1	**PSPACE**-hard Thm. 6 In **NEXPTIME** Thm. 6	**PSPACE**-hard Prop. 3 In **coNEXPTIME** Prop. 3	**PSPACE**-hard Prop. 4 In **coNEXPTIME** Prop. 4	**coNEXPTIME**-hard Prop. 5 In **co2NEXPTIME** Prop. 5

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.

accounting for the size of the problem is crucial because the runtime of a certain algorithm (operation) for the problem is measured with respect to the problem size. Notably, the size of an object varies in how it is encoded, e.g., a number can be encoded either in binary or in unary, which can affect the complexity of the problem. As an example, the unary encoding of 5 is "11111", and its binary encoding is "101".

Grounded Classical Planning A *grounded* classical planning problem is a tuple $\Pi = (\mathcal{D}, s_I, g)$ where $\mathcal{D} = (\mathcal{F}, \mathcal{A}, \alpha)$ is called the *domain* of Π. $\mathcal{F}$ is a (finite) set of *propositions*, $\mathcal{A}$ is a (finite) set of *action names* (or actions for short), and $\alpha : \mathcal{A} \to 2^{\mathcal{F}} \times 2^{\mathcal{F}} \times 2^{\mathcal{F}}$ is a function mapping each action $a \in \mathcal{A}$ to its precondition, add list, and delete list, written $\alpha(a) = (prec(a), add(a), del(a))$. $s_I \in 2^{\mathcal{F}}$ is the *initial state* of Π and $g \subseteq \mathcal{F}$ 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., $s \in 2^{\mathcal{F}}$. Applying an action $a \in \mathcal{A}$ in a state s will result in a new state s' with $s' = (s \setminus del(a)) \cup add(a)$. An action a is *applicable* in a state s if $prec(a) \subseteq s$. In other words, the precondition of a is satisfied in s. For convenience, we write $s \to_a s'$ 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 $\pi = \langle a_1 \cdots a_n \rangle$ ($n \in \mathbb{N}$), we write $s \to_\pi^* s'$ for some state s' to indicate that s' is obtained by applying π in s, that is, there exists a state sequence $\langle s_0 \cdots s_n \rangle$ such that $s_0 = s$, $s_n = s'$, and for each $1 \leq i \leq n$, $s_{i-1} \to_{a_i} s_i$. Consequently, a solution to a (grounded) classical planning problem is an action sequence π such that $s_I \to_\pi^* s'$ for some state s' and $g \subseteq s'$.

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 $\mathbf{p} \in \mathcal{R}$ is of the form $\mathbf{p} = P(v_1, \cdots, v_n)$ 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* $\varrho : \mathcal{V} \to \mathcal{O}$. More concretely, given a variable substitution function ϱ, grounding the predicate $\mathbf{p}$ according to ϱ results in the *grounded* predicate, written $\mathbf{p}[\![\varrho]\!]$, with $\mathbf{p}[\![\varrho]\!] = P(\varrho(v_1), \cdots, \varrho(v_n))$. In particular, a grounded predicate is equivalent to a proposition in the grounded classical planning formalism.

A lifted planning problem is again a tuple $\mathbf{\Pi} = (\mathcal{D}, s_I, g)$ with $\mathcal{D} = (\Sigma, \mathcal{A}, \alpha)$ being its domain. In the lifted setting, $\mathcal{A}$ is a set of *action schemas*. An action schema, $\mathbf{a} \in \mathcal{A}$, 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. α maps an action schema to its precondition, add list, and delete list, written $\alpha(\mathbf{a}) = (prec(\mathbf{a}), add(\mathbf{a}), del(\mathbf{a}))$, each of which is a set of *predicates* $P(v_{i_1}, \cdots, v_{i_j})$ such that $v_{i_r} \in \{v_1, \cdots, v_n\}$ for each $r \in \{1, \cdots, j\}$.

An action schema $\mathbf{a}$ can also be grounded into an *action* a in the grounded setting by a variable substitution function ϱ, written $a = \mathbf{a}[\![\varrho]\!]$. 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 $\mathbf{\Pi}$, respectively. A solution to $\mathbf{\Pi}$ is an *action* sequence $\pi = \langle a_1 \cdots a_n \rangle$ such that $s_I \to_\pi^* s'$ for some state s' with $g \subseteq s'$, and for each a_i with $1 \leq i \leq n$, there exist an action schema $\mathbf{a} \in \mathcal{A}$ and a variable substitution function ϱ such that $a_i = \mathbf{a}[\![\varrho]\!]$.

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

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 focus on one single problem instance.

Notably, one can obtain a *grounded* planning problem Π from a lifted one $\mathbf{\Pi}$ by grounding *every* predicate and action schema with *all* possible variable substitution functions, and the problem Π produced in such a way has the same solution set as $\mathbf{\Pi}$. One important remark is that $\|\Pi\|$ is exponential in $\|\mathbf{\Pi}\|$, that is, $\|\Pi\| = O(2^{\|\mathbf{\Pi}\|^q})$ 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 Π is a tuple $(\mathcal{D}, s_I, tn_I, g)$ with $\mathcal{D} = (\mathcal{F}, \mathcal{A}, \mathcal{C}, \mathcal{M}, \alpha)$ being its domain. A grounded HTN planning problem is an extension of a grounded classical one in the sense that $\mathcal{F}$, $\mathcal{A}$, α, s_I, and g are defined in the same way as their counterparts in the classical setting. An action $a \in \mathcal{A}$ in HTN planning is also called a *primitive task*. Two components, $\mathcal{C}$ and $\mathcal{M}$, which are not in the classical formalism, are the set of *compound tasks* and of *methods*, respectively. A method $(c, tn) \in \mathcal{M}$ *decomposes* a compound task $c \in \mathcal{C}$ into a so-called *task network* tn, which is essentially a partial order *multiset* 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 $\gamma : T \to \mathcal{A} \cup \mathcal{C}$ 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 $\varphi : T \to T'$ such that $\gamma(t) = \gamma'(\varphi(t))$ for any $t \in T$, and for any $t, t' \in T$, $(t, t') \in \prec$ *iff* $(\varphi(t), \varphi(t')) \in \prec'$. The last component tn_I in Π 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 $m = (c, tn^\dagger)$, written $tn \Rightarrow_m tn'$, if there exists an identifier $t \in T$ and a task network $tn^* = (T^*, \prec^*, \gamma^*)$ with $tn^* \cong tn^\dagger$ such that

1) $T^* \cap T = \emptyset$
2) $T' = (T \setminus \{t\}) \cup T^*$
3) $\gamma(t) = c$
4) $\gamma' = (\gamma \setminus \{(t, c)\}) \cup \gamma^*$
5) $\prec' = (\prec \setminus \prec_t) \cup \prec^* \cup \prec_\delta$ in which $\prec_t$ is the set of ordering constraints $\{(t', t) \mid (t', t) \in \prec\} \cup \{(t, t') \mid (t, t') \in \prec\}$, 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 $\{(t_1, t_2) \mid t_2 \in T^*, (t_1, t) \in \prec\} \cup \{(t_2, t_1) \mid t_2 \in T^*, (t, t_1) \in \prec\}$.

Further, let tn and tn' be two task networks and $\overline{m}$ a sequence of methods. We use $tn \Rightarrow^*_{\overline{m}} tn'$ to indicate that tn' is obtained from tn by applying $\overline{m}$.

Like classical planning, (grounded) HTN planning is also to find an action sequence (i.e., a plan) which turns s_I 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 π is a solution to an HTN planning problem Π if $s_I \Rightarrow^*_\pi s$ with $g \subseteq s$ for some state s, and there exists a task network $tn = (T, \prec, \gamma)$ such that $tn_I \Rightarrow^*_{\overline{m}} tn$ for some method sequence $\overline{m}$, and tn has a *linearization* $\overline{tn}$ that forms π. A linearization $\overline{tn} = \langle t_1 \cdots t_{|T|} \rangle$ of tn is a total order of T which respects $\prec$, and by $\overline{tn}$ forming π, we mean that $\pi = \langle \gamma(t_1) \cdots \gamma(t_{|T|}) \rangle$. For convenience, we use $\gamma(\overline{tn})$ to denote the task sequence formed by $\overline{tn}$. 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 $\mathbf{\Pi} = (\mathcal{D}, s_I, tn_I, g)$ with $\mathcal{D} = (\Sigma, \mathcal{A}, \mathcal{C}, \mathcal{M}, \alpha)$ being its domain where $\Sigma = (\mathcal{V}, \mathcal{O}, \mathcal{R})$, $\mathcal{A}$, and α are defined in the same way as that in lifted classical planning. Every action schema is also called a *primitive task schema*. $\mathcal{C}$ is now a set of *compound task schemas* and $\mathcal{M}$ a set of *method schemas*. A compound task schema $\mathbf{c} \in \mathcal{C}$ is simply a compound task name together with a tuple of variables. A method schema $\mathbf{m}$ is a tuple $(\mathbf{c}, \mathbf{tn})$ where $\mathbf{c}$ is a compound task schema and $\mathbf{tn}$ 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 γ maps each identifier to a *task schema*.

A task, task network, or method schema $\mathbf{x}$ can again be grounded by some variable substitution function $\varrho : \mathcal{V} \to \mathcal{O}$, written $\mathbf{x}[\![\varrho]\!]$. When grounding a task network schema $\mathbf{tn}$ with a substitution function ϱ, all task schemas in $\mathbf{tn}$ are grounded simultaneously by ϱ, and for any method schema $\mathbf{m} = (\mathbf{c}, \mathbf{tn})$ with $\mathbf{m}[\![\varrho]\!] = (\mathbf{c}[\![\varrho]\!], \mathbf{tn}[\![\varrho]\!])$. A grounded task schema and a grounded method schema are equivalent to a task and a method in the grounded setting, respectively.

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

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 $\mathbf{\Pi}$ be a lifted (classical or hierarchical) planning problem and Π its grounded counterpart. Then it holds that $\|\Pi\| = O(2^{\|\mathbf{\Pi}\|^q})$ 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 **P**.

In contrast, the plan verification problem in HTN planning is more computationally expensive. Previous works have already shown that it is already **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 **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 **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 **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 **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 **NP**-complete but not **PSPACE**-complete because it is **PSPACE**-complete to decide whether a word is in a context-sensitive language (CSL) (Immerman 1988; Szelepcsényi 1987) while the solution set of a grounded HTN problem is a CSL (Höller et al. 2014). It is because the language, denoted $\mathcal{L}_1$, of the solution set of a grounded HTN problem Π is different from the language of the word membership problem for CSLs, denoted $\mathcal{L}_2$. More specifically, $\mathcal{L}_1$ is $\{\pi \mid \pi \in \mathtt{sol}(\Pi)\}$ where $\mathtt{sol}(\Pi)$ is the solution set of Π whereas $\mathcal{L}_2$ is the set $\{(\omega, \Gamma) \mid \omega \in \Gamma\}$ in which Γ is a context-sensitive grammar and ω a word. That is, every element in $\mathcal{L}_2$ is a *tuple* of a word and a grammar, but every element in $\mathcal{L}_1$ is only a plan. $\mathcal{L}_2$ is **PSPACE**-complete while the complexity of $\mathcal{L}_1$ depends on Π.

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 **PSPACE**-hard.*

Proof. We will reduce from the *grounded* classical plan existence problem. Let $\Pi = (\mathcal{D}, s_I, g)$ with $\mathcal{D} = (\mathcal{F}, \mathcal{A}, \alpha)$ be a grounded classical planning problem. For convenience, we assume that $\mathcal{F} = \{p_1, \cdots, p_n\}$ with $n \in \mathbb{N}$. Our objective is thus to construct a lifted HTN planning problem $\mathbf{\Pi}$ and a plan π such that π is a solution to $\mathbf{\Pi}$ if and only if Π 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 $\mathcal{A}$, 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* such that each grounding of this schema represents a state. This compound task schema $\mathbf{c}$ is constructed as follows:

$$\mathbf{c} = \mathtt{State}(x_1, \cdots, x_n, v_0, v_1)$$

where $x_1 \cdots x_n, v_0, v_1$ are variables, and $n = |\mathcal{F}|$. Each x_i ($1 \leq i \leq n$) represent the respective proposition $p_i \in \mathcal{F}$. Our constructed lifted HTN problem contains only two objects, namely, 0 and 1, and hence, for any *grounded* version of $\mathbf{c}$, 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 $a \in \mathcal{A}$, we construct a method schema $\mathbf{m}_a$ decomposing the task schema $\mathtt{State}(x'_1, \cdots, x'_n, v_0, v_1)$ into a task network schema which contains solely one compound task schema $\mathtt{State}(x^*_1, \cdots, x^*_n, v_0, v_1)$ such that for all $1 \leq i \leq n$, 1) $x'_i = v_1$ if $p_i \in \mathit{prec}(a)$, 2) $x^*_i = v_0$ if $p_i \in \mathit{del}(a)$, 3) $x^*_i = v_1$ if $p_i \in \mathit{add}(a)$, 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 $\texttt{State}$, 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 $prec(a) \subseteq s$. Similarly, those x_i^*'s with $x_i^* = v_1$ (*resp.* $x_i^* = v_0$) 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:

$$c_I = \texttt{State}(y_1, \cdots, y_n, 0, 1)$$

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 π to be verified. To construct this plan, we first need to construct n extra *compound* task schemas $\texttt{P}_i(x)$'s, one for each $p_i \in \mathcal{F}$, and one method schema $\mathbf{m}_s$ which decomposes the task schema $\texttt{State}(x_1, \cdots, x_n, v_0, v_1)$ into the *total order* task network $\langle \texttt{P}_1(x_1) \cdots \texttt{P}_n(x_n) \rangle$. The purpose of these $\texttt{P}_i$'s and $\mathbf{m}_s$ is to *extract* each proposition p_i from the respective state. Each $\texttt{P}_i(x)$ can further be decomposed by *two* method schemas. One decomposes $\texttt{P}_i(x)$ into the task network consisting of only one *primitive* task schema $\texttt{ExistP}_i(x)$ that has neither precondition nor effects. The other method schema decomposes $\texttt{P}_i(x)$ into the primitive task schema $\texttt{NotCare}()$, which again has no precondition and effects. Each $\texttt{ExistP}_i(x)$ can be viewed as a certificate of whether the proposition p_i is in the respective state, namely, $\texttt{ExistP}_i(1)$ asserts the presence of p_i, and $\texttt{ExistP}_i(0)$ asserts its absence. $\texttt{NotCare}()$ simply means that we don't care whether the proposition is in the state. Finally, the plan π to be verified is the sequence $\langle a_1 \cdots a_n \rangle$ where for each $1 \leq i \leq n$, $a_i = \texttt{ExistP}_i(1)$ if $p_i \in g$, otherwise, $a_i = \texttt{NotCare}()$. The interpretation of the plan π 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 π' in the Π 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 π. □

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_I encoding s_I is $\texttt{State}(1, 0, 0, 0, 1)$ because s_I has only p_1. a_1 is the only action which is applicable in s_I. Thus, only $\mathbf{m}_{a_1}$ 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

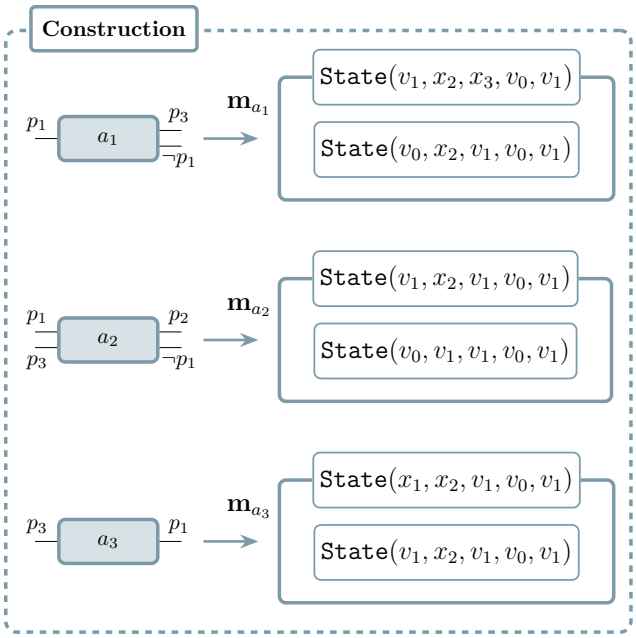

Figure 1: Encoding actions as method *schemas*.

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 $\texttt{State}(0, 0, 1, 0, 1)$ which encodes the state obtained by applying a_1 in the initial state.

The plan π to be verified that encodes the goal of the classical problem is $\langle t_1 \ t_2 \ t_3 \rangle$ where $t_2 = \texttt{ExistP}_2(1)$ and $t_1 = t_3 = \texttt{NotCare}()$. This plan is a solution to the lifted HTN problem. The decomposition resulting in π is to apply in sequence $\mathbf{m}_{a_1}$, $\mathbf{m}_{a_3}$, and $\mathbf{m}_{a_2}$, each with the corresponding grounding, to obtain the task $\texttt{State}(0, 1, 1, 0, 1)$, which can be further decomposed into π. 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 π shall be $\langle \texttt{ExistP}_1(1) \ \texttt{ExistP}_2(1) \ \texttt{NotCare}() \rangle$. Now this plan is *not* a solution to the HTN problem because no decomposition can ground x_1 in $\texttt{State}$ to 1.

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

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

Bounded Plan Existence

We now move on to discuss the complexity of the bounded (k-length) plan existence problem, which is to decide, given

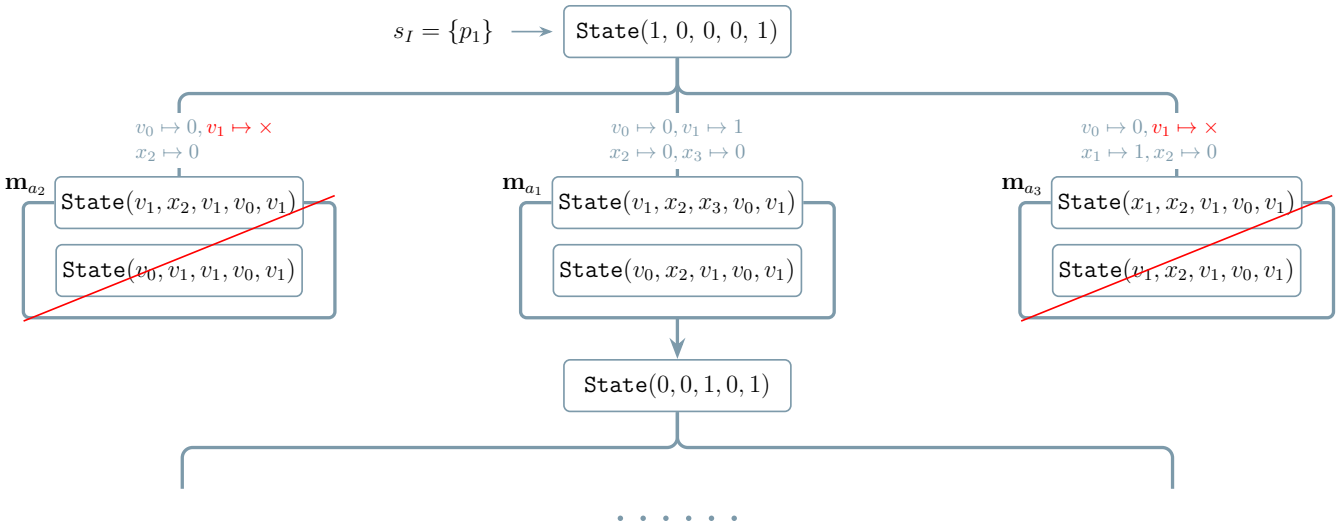

Figure 2: An example of using a decomposition hierarchy to simulate actions' executions in a classical planning problem.

a planning problem and a $k \in \mathbb{N}$, whether there is a solution plan π 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 π of length up to the bound k and then verify whether π is a solution to the given planning problem Π. Bearing this observation, one could notice that the complexity of the bounded plan existence problem is restricted by the formula:

$$T_V^N(\|\Pi\| + \|\pi\|) + T_G^N(\|k\|) \tag{1}$$

where T_V^N is a function that denotes the runtime with respect to the encoding size of Π and π for *verifying* whether π is a solution, and T_G^N denotes the runtime with respect to the encoding size of k for guessing π. The superscript N indicates that both operations are done non-deterministically.

Having Formula (1) in hand, one could observe that **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 $T_G^N(\|k\|)$ is exponential to $\|k\|$, and 2) verifying whether a plan is a solution to a planning problem is in **NP** with respect to the encoding size of the plan and the planning problem, which further ensures that $T_V^N(\|\Pi\| + \|\pi\|)$ 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 **NEXPTIME**-membership. In fact, the tight bound for the grounded representation have been proved earlier (Erol, Nau, and Subrahmanian 1991; Bylander 1994), which is **PSPACE**-complete, making it as hard as its unbounded version (Bylander 1994). **NEXPTIME**-completeness for the lifted setting has also been proved by Erol, Nau, and Subrahmanian (1991) whereas its unbounded counterpart is **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 **NEXPTIME**-complete.*

Proof. Membership can be obtained by recognizing the fact that grounded HTN planning satisfies the two properties that assert **NEXPTIME**-membership. For hardness, we reduce from the *grounded acyclic* HTN plan existence problem. The basis for the reduction is the result by Behnke 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 $T_V^N(\|\Pi\| + \|\pi\|)$ 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 **NEXPTIME**-hard and in 2**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 exponentionally.

Bearing this scenario, Bäckström and Jonsson (2011) investigated the k-length plan existence problem from a dif-

ferent aspect where they developed its complexity with respect to the magnitude of the bound. This is done by assuming that the bound is encoded in *unary*. The authors studied this for *finite functional planning* (FFP) and proved its **NP**-completeness. They further justified that a *grounded* classical planning problem can be reduced to an FFP problem in poly-time (Bäckström and Jonsson 2011, Prop. 1) (note that this does *not* hold for the lifted formalism), and hence, **NP**-completeness also holds in grounded classical planning.

We now extend the result by Bäckström and Jonsson to lifted classical planning and grounded and lifted HTN planning. We first note that when k is given in unary, the term $T_G^N(\|k\|)$ in Formula (1) becomes a polynomial. Hence, one could immediately recognize that **NP**-membership holds for *any* planning formalism in which $T_V^N(\|\Pi\| + \|\pi\|)$ 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 $T_V^N(\|\Pi\| + \|\pi\|)$ 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 $\Pi = (\mathcal{D}, s_I, g)$ 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 $\langle a\ c \rangle$. 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 **NEXPTIME**.*

Proof. Thm. 2 implies that the term $T_V^N(\|\Pi\| + \|\pi\|)$ in Formula (1) is exponential to $\|k\|$ due to the unary encoding. As a result, **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 $\texttt{Pred}_i(x)$, 2) for each proposition $p_i \in \mathcal{F}$, the respective primitive task schema $\texttt{ExistP}_i(x)$ now has a single positive effect $\texttt{Pred}_i(x)$, and 3) the goal of the lifted HTN problem is $\{\texttt{Pred}_i(1) \mid 1 \leq i \leq n, p_i \in g\}$.

Every proposition $\texttt{Pred}_i(1)$ can only be obtained from the respective primitive task $\texttt{ExistP}_i(1)$. Hence, the lifted HTN problem has a solution *iff* there exists a decomposition hierarchy that results in a plan containing all $\texttt{ExistP}_i(1)$'s with $p_i \in g$, 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 Σ_2^p-complete (Sreedharan, Bercher, and Kambhampati 2022). The second one is concerned with whether a plan can be further optimized by removing some redundant actions from it, which is **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 **P**, **NP**-complete, or Σ_2^p-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 π 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 π is an optimal one is identical to asking whether there exist no solution plans of length smaller or equal to $|\pi| - 1$ with $|\pi| - 1$ 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 **coNP**-complete. For lifted HTN planning, this problem is **PSPACE**-hard and in **coNEXPTIME**.*

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 Π, a solution plan π to Π, and a bound k, whether the length $|\pi^*|$ of an optimal solution π^* to Π satisfies $|\pi| < |\pi^*| + k$. In other words, we want to verify whether the length of π is *not* larger than the length of an optimal solution by the bound k. (Note that both $|\pi^*|$ and π^* 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 π' to Π such that $|\pi| - k > |\pi'|$. For if such a π' exists, we have $|\pi^*| \leq |\pi'|$ because π^* 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 π 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 Π, and *two* numbers k_π and k where k_π is the length of some solution, we want to decide whether there exist *no* solution plans π' to Π of length k' such that $k_\pi - k' > k$. 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 π with $|\pi|$ steps (in some domains, the modeler might be aware that the solution π 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 2**NEXPTIME** depending on planning formalisms whereas it decreases to the range from **NP**-complete to **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

Alford, R.; Bercher, P.; and Aha, D. 2015a. Tight Bounds for HTN planning with Task Insertion. In *IJCAI 2015*, 1502–1508. AAAI Press.

Alford, R.; Bercher, P.; and Aha, D. W. 2015b. Tight Bounds for HTN Planning. In *ICAPS 2015*, 7–15. AAAI.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2014. On the Feasibility of Planning Graph Style Heuristics for HTN Planning. In *ICAPS 2014*, 2–10. AAAI.

Bachor, P.; and Behnke, G. 2024. Learning Planning Domains from Non-Redundant Fully-Observed Traces: Theoretical Foundations and Complexity Analysis. In *AAAI 2024*. AAAI.

Bäckström, C.; and Jonsson, P. 2011. All PSPACE-Complete Planning Problems Are Equal but Some Are More Equal than Others. In *SoCS 2011*, 10–17. AAAI.

Barták, R.; Ondrcková, S.; Behnke, G.; and Bercher, P. 2021. On the Verification of Totally-Ordered HTN Plans. In *ICTAI 2021*, 263–267. IEEE.

Behnke, G.; Höller, D.; Bercher, P.; and Biundo, S. 2016. Change the Plan – How Hard Can That Be? In *ICAPS 2016*, 38–46. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the Complexity of HTN Plan Verification and its Implications for Plan Recognition. In *ICAPS 2015*, 25–33. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2019. Finding Optimal Solutions in HTN Planning – A SAT-based Approach. In *IJCAI 2019*, 5500–5508. IJCAI.

Behnke, G.; and Speck, D. 2021. Symbolic Search for Optimal Total-Order HTN Planning. In *AAAI 2021*, 11744–11754. AAAI.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *IJCAI 2019*, 6267–6275. IJCAI.

Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An Admissible HTN Planning Heuristic. In *IJCAI 2017*, 480–488. IJCAI.

Bercher, P.; Lin, S.; and Alford, R. 2022. Tight Bounds for Hybrid Planning. In *IJCAI-ECAI 2022*, 4597–4605. IJCAI.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *AIJ*, 94: 165–204.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Ann. Math. Artif. Intell.*, 18: 69–93.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1991. Complexity, Decidability and Undecidability Results for Domain-Independent Planning: A Detailed Analysis. Technical Report CS-TR-2797, UMIACS-TR-91-154, SRC-TR-91-96, University of Maryland, College Park, Maryland, USA.

Fink, E.; and Yang, Q. 1992. Formalizing Plan Justifications. In *CSCSI 1992*, 9–14. ACM.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *IJCAI 2011*, 1955–1961. IJCAI.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning – Theory and Practice*. Elsevier.

Helmert, M. 2006. New Complexity Results for Classical Planning Benchmarks. In *ICAPS 2006*, 52–62. AAAI.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *ECAI 2014*, 447–452. IOS.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI 2004*, 294–301. IEEE.

Immerman, N. 1988. Nondeterministic Space is Closed under Complementation. *SIAM J. Comput.*, 17: 935–938.

Karpas, E.; and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. In *IJCAI 2009*, 1728–1733. IJCAI.

Lin, S.; Behnke, G.; Ondrčková, S.; Barták, R.; and Bercher, P. 2023. On Total-Order HTN Plan Verification with Method Preconditions – An Extension of the CYK Parsing Algorithm. In *AAAI 2023*, 12041–12048. AAAI.

Lin, S.; and Bercher, P. 2021. Change the World – How Hard Can that Be? On the Computational Complexity of Fixing Planning Models. In *IJCAI 2021*, 4152–4159. IJCAI.

Lin, S.; and Bercher, P. 2023. Was Fixing this *Really* That Hard? On the Complexity of Correcting HTN Domains. In *AAAI 2023*, 12032–12040. AAAI.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In *AAAI 2023*, 12022–12031. AAAI.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *K-CAP 2017*, 1–8. ACM.

Nakhost, H.; and Müller, M. 2010. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement. In *ICAPS 2010*, 121–128. AAAI.

Olz, C.; and Bercher, P. 2019. Eliminating Redundant Actions in Partially Ordered Plans – A Complexity Analysis. In *ICAPS 2019*, 310–319. AAAI.

Olz, C.; and Bercher, P. 2023. A Look-Ahead Technique for Search-Based HTN Planning: Reducing the Branching Factor by Identifying Inevitable Task Refinements. In *SoCS 2023*, 65–73. AAAI.

Olz, C.; Biundo, S.; and Bercher, P. 2021. Revealing Hidden Preconditions and Effects of Compound HTN Planning Tasks – A Complexity Analysis. In *AAAI 2021*, 11903–11912. AAAI.

Pantucková, K.; and Barták, R. 2023. Using Earley Parser for Recognizing Totally Ordered Hierarchical Plans. In *ECAI 2023*, 1819–1826. IOS.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-Based Heuristics for Cost-Optimal Planning. In *ICAPS 2014*, 226–234. AAAI.

Sreedharan, S.; Bercher, P.; and Kambhampati, S. 2022. On the Computational Complexity of Model Reconciliations. In *IJCAI 2022*, 4657–4664. IJCAI.

Szelepcsényi, R. 1987. The Method of Forcing for Nondeterministic Automata. *Bull. EATCS*, 33: 96–99.