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 at the bounded plan existence problem. We do this for standard classical planning and hierarchical task network (HTN) planning and each for the grounded and the lifted representation. Whereas bounded plan existence complexity is known for classical planning, it has not been studied yet for HTN planning. For plan verification, results were available for both formalisms except the lifted representation of HTN planning. We will thus present the lower bound and the upper bound of the complexity of plan verification in lifted HTN planning and provide novel 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 computational complexity concerning the optimality of a given plan, i.e., answering the question whether such a plan is optimal, 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 2015) which is to decide whether a planning problem has a solution. In contrast, the number of research endeavors on the complexity of finding an *optimal* plan is relatively small. Despite that many approaches for finding optimal plans have been developed for both classical planning (Karpas and Domshlak 2009; Pommerening et al. 2014) and hierarchical planning (Bercher et al. 2017; Behnke, Höller, and Biundo 2019; Behnke and Speck 2021), the complexity results only exist in the classical setting but not in the hierarchical one.

We will discuss the complexity of several problems centered at the *bounded* plan existence problem (which is a standard way of framing the problem of finding an optimal solution as a decision problem). Our discussion starts 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 the lower bound and the 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 start by presenting the notations that will be used throughout the paper together with 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, accounting for the size of the problem is crucial because the runtime of a certain algorithm (operation) for the problem

		Plan Verification	k-length Plan Existence		Plan Optimality Verification	Bounded Plan Optimality Verification	
			k in binary	k in unary		plan given	only plan length given
Classical	*ground*	In **P**	**PSPACE**-complete (Bylander 1994)	**NP**-complete (Bäckström and Jonsson 2011)	**coNP**-complete Prop. 3	**coNP**-complete Prop. 4	**PSPACE**-complete Prop. 5
	lifted	In **P**	**NEXPTIME**-complete (Erol, Nau, and Subrahmanian 1991)	**NP**-complete Thm. 4	**coNP**-complete Prop. 3	**coNP**-complete Prop. 4	**coNEXPTIME**-complete Prop. 5
Hierarchical	*ground*	**NP**-complete Prop. 2	**NEXPTIME**-complete Thm. 3	**NP**-complete Thm. 5	**coNP**-complete Prop. 3	**coNP**-complete Prop. 4	**coNEXPTIME**-complete Prop. 5
	lifted	**PSPACE**-hard Thm. 1, Cor. 1 In **NEXPTIME** Thm. 2	**NEXPTIME**-complete Thm. 3	**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**-complete Prop. 5

Table 1: Summary of the complexity results and the respective theorems. Note that we demand here that a solution to an HTN planning problem is an *action sequence*, which is different from the standard definition of solutions where a solution is a primitive task network. The plan optimality verification problem and the bounded plan optimality verification problem with the plan given explicitly are semantically equivalent, and they are the *complement* of the bounded plan existence problem with the bound given in unary. The bounded plan optimality verification problem with only the plan length given is the complement of the bounded plan existence problem. As a special case, plan optimality verification with only the length of a plan being given is equivalent to setting the bound to zero in the bounded plan optimality verification (where the plan is also not given explicitly), and hence it is also the complement of the bounded plan existence problem (cf. Prop. 6). **NP**-completeness of plan verification in grounded HTN planning was proved (Behnke, Höller, and Biundo 2015). We show that it holds even in a severely restricted case.

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 $\mathcal{D} = (F, \mathcal{A}, \alpha)$ is called the *domain* of Π. F is a (finite) set of *propositions*, $\mathcal{A}$ is a (finite) set of *action names* (or actions for short), and $\alpha : \mathcal{A} \to 2^F \times 2^F \times 2^F$ is a function mapping each action $a \in \mathcal{A}$ to its precondition, add list, and delete list, written $\alpha(a) = (prec(a), add(a), del(a))$. $s_I \in 2^F$ is the *initial state* of Π and $g \subseteq F$ 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., $s \in 2^F$. 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* $\Sigma = (\mathcal{V}, \mathcal{O}, \mathcal{R})$ where $\mathcal{V}$ 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 $n \in \mathbb{N}$ 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, and 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 predicated 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 $a = \mathbf{a}[\![\varrho]\!]$. Notably, when grounding an action schema, all predicates in its precondition, add list, and delete list are grounded simultaneously by the same variable 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 $a_i = \mathbf{a}[\![\varrho]\!]$.

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 $\mathcal{D} = (F, \mathcal{A}, \mathcal{C}, \mathcal{M}, \alpha)$ being its domain. A grounded HTN planning problem is an extension of a grounded classical one in the sense that 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 $tn = (T, \prec, \gamma)$ 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) $\gamma(t) = c$, 3) $T' = (T \setminus \{t\}) \cup T^*$, 4) $\gamma' = (\gamma \setminus \{(t, c)\}) \cup \gamma^*$, and 5) $\prec' = (\prec \setminus \prec_t) \cup \prec^* \cup \prec_\delta$ with $\prec_t = \{(t', t) \mid (t', t) \in \prec\} \cup \{(t, t') \mid (t, t') \in \prec\}$, i.e., $\prec_t$ is the set of all ordering constraints in tn that are associated with t, and $\prec_\delta = \{(t_1, t_2) \mid t_2 \in T^*, (t_1, t) \in \prec\} \cup \{(t_2, t_1) \mid t_2 \in T^*, (t, t_1) \in \prec\}$, i.e., $\prec_\delta$ specifies the position of tn^* in tn' with respect to the task t replaced by it. 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

Having presented all planning formalisms involved in this paper, we move on to discuss the complexity results for the plan verification problem, 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 def-

inition of solutions being task networks that possess some executable linearization, whereas we define such a linearization as the solution itself. In fact, even with our solution criteria, HTN plan verification is still **NP**-complete in grounded HTN planning (Behnke, Höller, and Biundo 2015, Thm. 2).

The existing hardness proof (Behnke, Höller, and Biundo 2015) for the verification problem (in the context of our definition of solutions) relies on finding a decomposition hierarchy, i.e., hardness of the problem adheres to decomposition that results in the given plan. We can further improve this result by showing that **NP**-hardness already holds even if the initial task network is *primitive*. This follows trivially from the fact that it is **NP**-complete to decide whether an action sequence is a linearization of a partial order task network (Lin and Bercher 2023).

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

Notably, as a special case, the plan verification problem in the context of *total order* (TO) HTN planning is polytime decidable (Behnke, Höller, and Biundo 2015). A TOHTN planning problem is such that the initial task network is totally ordered, and every method decomposes 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, and many theoretical investigations into properties of TOHTN planning have been made which have great potential to be utilized to solve TO problems more efficiently (Olz, Biundo, and Bercher 2021). The poly-time decidability of TOHTN plan verification 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 in TOHTN planning. Bearing this connection, many efficient TOHTN plan verifiers (Barták et al. 2021; Lin et al. 2023) have been developed by exploiting CFG parsers.

Now 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. Concretely, we will show that plan verification is already **PSPACE**-hard even for lifted TOHTN planning.

Theorem 1. *The plan verification problem in lifted TOHTN planning is **PSPACE**-hard.*

Proof. We reduce from the plan existence problem in *grounded* classical planning. Suppose $\Pi = (\mathcal{D}, s_I, g)$ with $\mathcal{D} = (F, \mathcal{A}, \alpha)$ is a grounded classical planning problem. For convenience, we assume that, without loss of generality, $F = \{p_1, \cdots, p_n\}$ with $n \in \mathbb{N}$ and $g = \{p_{i_1}, \cdots, p_{i_j}\}$. The lifted HTN planning problem we are to construct has only two objects, namely, 0 and 1. At the central of the reduction is the compound task schema $\mathbf{c}$ which is of the form

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

with $n = |F|$. Each x_i with $1 \leq i \leq n$ represents the corresponding $p_i \in F$. Thus, a *grounded* version of the schema $\mathbf{c}$ encodes a *state*. Our construction will ensure that v_0 is *always* grounded to 0 and v_1 to 1 in decomposition, which is useful for simulating state transitions (and which can be done by the construction of the initial task network). More concretely, we will construct method schemas that decompose $\mathbf{c}$ to encode actions (in the given classical problem). For each action $a \in \mathcal{A}$, we construct a method schema $\mathbf{m}_a$ which decomposes the task schema

$$\texttt{State}(x'_1, \cdots, x'_n, v_0, v_1)$$

into another one $\texttt{State}(x^*_1, \cdots, x^*_n, v_0, v_1)$ such that for all $1 \leq i \leq n$, $x'_i = v_1$ if $p_i \in \mathit{prec}(a)$, $x^*_i = v_0$ if $p_i \in \mathit{del}(a)$, $x^*_i = v_1$ if $p_i \in \mathit{add}(a)$, and $x'_i = x^*_i$ if none of the previous holds. Intuitively, all x'_i's with $x'_i = v_1$ together enforce that the precondition of a must hold (because we will ensure that v_1 can only be grounded to 1), and similarly, those x^*_i's with $x^*_i = v_1$ (*resp.* $x^*_i = v_0$) enforce that the respective propositions should be added (*resp.* deleted).

As an example for the construction, 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 precondition and effects of each action are depicted in the most left column of Fig. 1 (inside the box labeled with construction). Those on the left side of an action are preconditions, and those on the right are effects. Each effect with a negation symbol in front of it (e.g., $\neg p_1$ in the action a_1) is in the delete list of the respective action, otherwise it is in the add list. On the right side of each action is the corresponding method schema that encodes it. For instance, the method schema with respect to a_1 decomposes the task schema

$$\texttt{State}(v_1, x_2, x_3, v_0, v_1)$$

into $\texttt{State}(v_0, x_2, v_1, v_0, v_1)$. The first v_1 in the decomposed task schema is changed to v_0 because the proposition p_1 is in the delete list of a_1, and x_3 becomes v_1 because p_3 is in the add list. x_2 is unchanged because the execution of a_2 will not affect p_2.

Having simulated each action, we now encode the initial state s_I of the classical problem and enforce that the parameters v_0 and v_1 of $\mathbf{c}$ can only be grounded to 0 and 1 in decomposition, respectively. This is done by the construction of the initial task network of the HTN problem, which consists solely of one *grounded* compound task:

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

where for each i with $1 \leq i \leq n$, $y_i = 1$ if the respective p_i is in s_I, otherwise, $y_i = 0$. By letting $v_0 = 0$ and $v_1 = 1$ in the initial task, we enforce that the values of those two variables cannot be changed in decomposition, because in each method schema we construct, the variables v_0 and v_1 are always inherited down from the task schema to be decomposed to the subtask schema. Hence, the correctness of our construction for simulating executions of actions follows.

Recall the classical problem presented in Fig. 1, the right side of the figure illustrates how the decomposition hierarchy simulates the actions' executions, using our construction of method schemas. Concretely, assume that the initial state

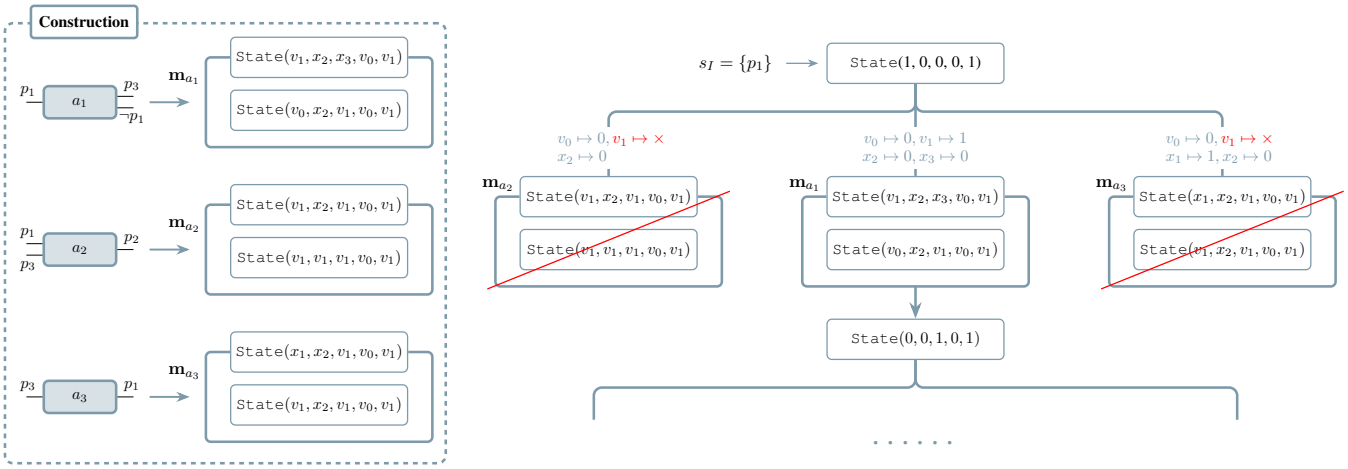

Figure 1: An example of using a decomposition hierarchy to simulate actions' executions in a classical planning problem. The left side shows how each action is encoded by a method schema, and the right side shows the decomposition.

of the classical problem is $\{p_1\}$. This results in the *grounded* initial compound task:

$$\texttt{State}(1, 0, 0, 0, 1)$$

because only p_1 is true in the state. a_1 is the only action that is applicable in the initial state. As a consequence, only the method schema $\mathbf{m}_{a_1}$ has a corresponding grounded version that can decompose the initial compound task. Specifically, for the action a_2, since it requires p_3 which is not in the initial state, 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. The decomposition of the initial compound task results in a new compound task $\texttt{State}(0, 0, 1, 0, 1)$ which encodes the state obtained by applying a_1 in the initial state.

Lastly, we will encode the goal description and the criterion that the goal must be satisfied. To this end, we first construct a method schema which decomposes the task schema $\texttt{State}(x_1, \cdots, x_n, v_0, v_1)$ into a *sequence* $\langle \mathbf{c}'_1 \cdots \mathbf{c}'_n \rangle$ of *compound* task schemas with

$$\mathbf{c}'_i = \texttt{P}_i(x_i)$$

for each $1 \leq i \leq n$. The compound task schema $\texttt{P}_i(x)$ can either be decomposed into an *action* schema

$$\mathbf{a}_i = \texttt{ValueP}_i(x)$$

or into an empty task network. The precondition, positive effects, and negative effects of $\mathbf{a}_i$ are all empty. We can view a grounded version of $\mathbf{a}_i$ as an assertion of the value of the proposition p_i in a state. If $\mathbf{a}_i$ is grounded by letting $x = 1$, it means that p_i holds in the respective state, and vice versa.

The plan to be verified should be $\langle a_{i_1}, \cdots, a_{i_j} \rangle$ where $a_{i_k} = \texttt{ValueP}_{i_k}(1)$ for each $1 \leq k \leq j$ (recall that each p_{i_k} is a proposition in the goal). This is to say that for each proposition in the goal, its truth value must be asserted. Since each task schema $\texttt{P}_i(x)$ can be decomposed into an empty task network no matter what object the variable x is grounded to, we can ensure that any action that is not in the constructed plan can be easily eliminated. Thus, the classical planning problem has a solution *iff* the constructed plan is a solution to the HTN planning problem. □

A by product of the presented proof is that it shows the expressive power of a decomposition hierarchy, i.e., a decomposition hierarchy can carry out certain semantics. For instance, Fig. 1 shows how the semantics of actions is encoded by a decomposition hierarchy. Thus, we also believe that the proof here can serve as a counter-argument for the *incorrect* commonsense that decomposition hierarchies in hierarchical planning can only serve as a guidance for finding plans but do not carry any information (semantics).

As a simple corollary of Thm. 1, **PSPACE**-hardness holds as well in general lifted HTN planning.

Corollary 1. *The plan verification problem in lifted HTN planning is **PSPACE**-hard.*

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**.*

In fact, one can recognize that plan verification for lifted TOHTN planning is actually in **EXPTIME**. This is because the grounded TOHTN plan verification problem is in **P**, and hence, after grounding a lifted problem $\mathbf{\Pi}$ into a grounded one Π, we can verify *deterministically* whether a given plan is a solution in polynomial time with respect to $\|\Pi\|$, which is exponential time with respect to $\|\mathbf{\Pi}\|$, e.g., by using the CYK algorithm for TOHTN problems (Lin et al. 2023).

Bounded Plan Existence

We now move on to discuss the complexity of the bounded (k-length) plan existence problem, which is to decide, given 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 non-deterministically by a two-steps procedure independent of any planning formalism, namely, we can first guess a plan of length up to the bound k and then verify whether this plan is a solution to the given planning problem. Further, notice that since the bound k is normally encoded in *binary*, guessing a plan that is bounded in length by k would thus require exponential time complexity. On top of this observation, we can obtain two properties which assert **NEXPTIME**-membership of the bounded plan existence problem for a planning formalism if these two properties hold in that formalism. Concretely, the two properties are as follows:

1) An action can be encoded in polynomial bits with respect to the size of the planning problem. This thus implies that guessing a plan of length up to the bound k can be done in time $O(2^{\|k\|^q})$ for some constant $q \in \mathbb{N}$.
2) Verifying whether a plan is a solution to the planning problem can be done non-deterministically in exponential time with respect to the encoding size of the planning problem and of the plan, i.e., the plan verification problem is in **NEXPTIME**.

These two properties together ensure that guessing and verifying a plan can be done in exponential time.

As a result, the bounded plan existence problem in both classical and HTN planning, including both the grounded and the lifted setting, is in **NEXPTIME**. The problem is actually **PSPACE**-complete (Erol, Nau, and Subrahmanian 1991; Bylander 1994) in *grounded* classical planning (note that this is *not* a contradiction because **PSPACE** is a subset of **NEXPTIME**), making it as hard as its unbounded version (Bylander 1994). **NEXPTIME**-completeness of the problem in lifted classical planning has also been proved by Erol, Nau, and Subrahmanian (1991), and its unbounded counterpart in the lifted setting is **EXPSPACE**-complete. For HTN planning, we will show that **NEXPTIME**-hardness holds as well in both the grounded and lifted HTN planning.

Theorem 3. *The k-length (bounded) plan existence problem for both grounded and lifted HTN planning is **NEXPTIME**-complete.*

Proof. Membership follows from Prop. 2 and Thm. 2. For hardness, we first show that the problem is **NEXPTIME**-hard in the grounded setting. We reduce from the *grounded acyclic* HTN plan existence problem. The basis for such a reduction is the fact shown by Behnke et al. (2016) that for any acyclic HTN planning problem, the length of a solution is bounded by an exponential number k^* with

$$k^* = \left(\max_{(c,(T,\prec,\gamma)) \in \mathcal{M}} |T| \right)^{|\mathcal{A}|}$$

Hence, by letting $k = k^*$, deciding whether an acyclic HTN planning has a solution is equivalent to deciding whether that acyclic HTN problem has a solution bounded in size by k. **NEXPTIME**-hardness of the bounded plan existence problem in grounded HTN planning follows immediately. Since a grounded HTN problem can be viewed as a special case of a lifted one, it follows that **NEXPTIME**-hardness holds as well in lifted HTN planning. □

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 encoding size of k is growing exponentially while its magnitude is growing polynomially. 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.

Bearing this scenario, Bäckström and Jonsson (2011) investigated the k-length plan existence problem from a different 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 HTN planning. Notice first that when k is given in unary, we can again identify two properties of a planning formalism which assert **NP**-membership and which are similar to the previous two that assert **NEXPTIME**-membership. Concretely, for any planning formalism, its k-length plan existence problem with k given in unary is in **NP** if the formalism holds the following two properties: 1) a plan step can be encoded in polynomial bits, and 2) the plan verification problem for the formalism is in **NP**. This is because the first property now implies that guessing a plan of length up to the bound k can be done in polynomial time with respect to the size of the planning problem and k, due to the unary encoding of k. Consequently, the time complexity of the entire guess-and-verify procedure can be done in polynomial time.

Hence, **NP**-membership in lifted classical planning and grounded HTN planning follows immediately because these formalisms preserve the two properties. In particular, **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 will prove **NP**-hardness for grounded HTN planning (and hence **NP**-completeness). Our proof relies on the reduction proposed by Erol, Hendler, and Nau (Erol, Hendler, and Nau 1996) from a grounded classical planning problem to a *regular* TOHTN problem[1] (which thus shows

[1] A regular HTN problem is such that a compound task can only occur at the last place in a method, i.e., all other tasks are ordered before it.

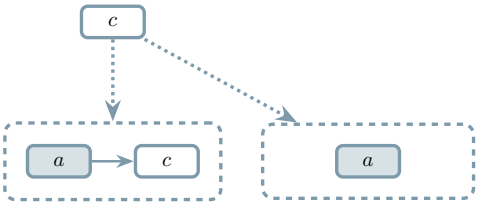

Figure 2: Simulating a grounded classical planning problem.

that total order regular HTN problems are **PSPACE**-hard).

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

Proof. Membership for both the grounded and lifted formalisms has been obtained. We will first show **NP**-hardness for the grounded setting, which thus implies **NP**-hardness for the lifted one.

We reduce from the k-length plan existence problem for *grounded classical* planning with k given in unary. Given a grounded classical planning problem $\Pi = (\mathcal{D}, s_I, g)$ with $\mathcal{D} = (F, \mathcal{A}, \alpha)$ and a $k \in \mathbb{N}$ in unary, we first simulate the classical problem by an HTN problem, using the construction proposed by Erol, Hendler, and Nau (Erol, Hendler, and Nau 1996). The HTN planning problem has the same action set as the classical one and solely one compound task c (which thus also serves as the initial task network). For each $a \in \mathcal{A}$, we construct two methods m_1 and m_2 with $m_1 = (c, (\{t\}, \emptyset, \{(t, a)\}))$ and

$$m_2 = (c, (\{t_1, t_2\}, \{(t_1, t_2)\}, \{(t_1, a), (t_2, c)\}))$$

An illustration of this construction is shown in Fig. 2. It simulates selections of actions in finding a solution to the classical planning 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 k (in unary). □

Unfortunately, **NP**-membership does *not* hold in lifted HTN planning because the plan verification problem in lifted HTN planning is **PSPACE**-hard.

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. Membership: Membership can be obtained by first guessing a plan of length up to k in poly-time (because k is encoded in unary) and then verifying non-deterministically whether it is a solution to a lifted HTN problem in exponential time (cf. Thm. 2).

Hardness: We again reduce from the grounded classical plan existence problem. The construction of the lifted HTN planning problem is identical to the one presented in the proof for Thm. 1 except few changes. More specifically, for each action schema $\texttt{ValueP}_i(x)$, we construct a predicate $\texttt{Prop}_i(x)$ as its positive effect, and the goal of the lifted HTN problem is $\{\texttt{Prop}_{i_j}(1) \mid p_{i_j} \in g\}$ where g is the goal description of the grounded classical planning problem. These modifications thus encode the solution criteria for the grounded classical planning problem and can replace the constructed plan that is to be verified in the proof for Thm. 1. Lastly, notice that any solution plan to the constructed lifted HTN planning problem is of length at most $|F|$ (where F is the proposition set of the given grounded classical planning problem). We can simply let $k = |F|$, and hence, by construction, the classical planning problem has a solution *iff* the constructed lifted HTN planning problem has one which is of length smaller or equal to k. □

Verification of Plan Optimality

Lastly, we discuss the problem of plan optimality verification, which is to decide, given a planning problem and a plan, whether there exist *no* other solution plans of length smaller than that of the given one. Many tasks of great importance are centered on plan optimality verification, for instance, the task of *model reconciliation* and of *plan post-optimization*. The former 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, and this task is Σ_2^p-complete (Sreedharan, Bercher, and Kambhampati 2022). The latter one is concerned with whether a plan can be further optimized by removing some redundant actions in it, and it is **NP**-complete in both classical planning (Fink and Yang 1992) and POCL planning (Olz and Bercher 2019).

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 representations, and grounded HTN planning is in **coNP**-complete, and it is in **coNEXPTIME** and is **PSPACE**-hard for lifted HTN planning.*

PSPACE-hardness in lifted HTN planning is due to the fact that **PSPACE** = **coPSPACE** (Arora and Barak 2009).

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.)

In spite of the fact that the bounded optimality verification problem describes a scenario 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 version of the bounded plan optimality verification problem, since we replace the given plan with a number, one could recognize that its complexity is 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.*

As a special case, 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 length of a plan is given is the complement of the bounded plan existence problem (with the bound given in binary).*

Conclusion and Extension

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 the bounded plan existence problem, its complexity ranges from **PSPACE**-complete to **NEXPTIME**-complete depending on planning formalisms, and the complexity decreases when the bound is encoded in unary. The problem of (bounded) plan optimality verification is the complement of bounded plan existence with the bound given in unary if the plan to be verified is explicitly given, and it is the complement of the bounded plan existence problem with the bound given in binary if only the length of the plan is given.

Extension In practice, an optimal solution usually refers to a plan of a minimal *cost*. That is, each action (in a planning problem) has a certain cost, and we want to find a solution plan which has an optimal cost (the cost of a plan is the sum of the cost of each action in it). The bounded plan existence problem studied in the paper can be viewed as a special case of this task where each action has cost one. Thus, the complexity results presented here naturally serve as a lower bound. In fact, the same upper bound also holds because for finding a cost optimal plan, we can again guess a plan up to a certain cost and then verify whether the guessed plan is a solution.

References

Alford, R.; Bercher, P.; and Aha, D. W. 2015. Tight Bounds for HTN Planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling, 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 *Proceedings of the 24th International Conference on Automated Planning and Scheduling, ICAPS 2014*, 2–10. AAAI.

Arora, S.; and Barak, B. 2009. *Computational Complexity – A Modern Approach*. Cambridge University Press.

Bäckström, C.; and Jonsson, P. 2011. All PSPACE-Complete Planning Problems Are Equal but Some Are More Equal than Others. In *Proceedings of the 4th Annual Symposium on Combinatorial Search, SoCS 2011*, 10 – 17. AAAI.

Barták, R.; Ondrcková, S.; Behnke, G.; and Bercher, P. 2021. On the Verification of Totally-Ordered HTN Plans. In *Proceedings of the 33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021*, 263–267. IEEE.

Behnke, G.; Höller, D.; Bercher, P.; and Biundo, S. 2016. Change the Plan - How Hard Can That Be? In *Proceedings*

of the 26th International Conference on Automated Planning and Scheduling, 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 *Proceedings of the 25th International Conference on Automated Planning and Scheduling, ICAPS 2015*, 25–33. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2019. Finding Optimal Solutions in HTN Planning - A SAT-based Approach. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, 5500–5508. IJCAI.

Behnke, G.; and Speck, D. 2021. Symbolic Search for Optimal Total-Order HTN Planning. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, 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 *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, 6267–6275. IJCAI.

Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An Admissible HTN Planning Heuristic. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI 2017*, 480–488. IJCAI.

Bercher, P.; Lin, S.; and Alford, R. 2022. Tight Bounds for Hybrid Planning. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI 2022*, 4597–4605. IJCAI.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 94(1-2): 165–204.

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

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 *Proceedings of the 9th Conference of the Canadian Society for Computational Studies of Intelligence, CSCSI 1992*, 9–14. ACM.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, 1955–1961. 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 *Proceedings of the 16th International Conference on Automated Planning and Scheduling, ICAPS 2006*, 52–62. AAAI.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *Proceedings of the 21st European Conference on Artificial Intelligence, ECAI 2014*, 447–452. IOS.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2004*, 294–301. IEEE.

Karpas, E.; and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, 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 *Proceedings of the 37th AAAI Conference on Artificial Intelligence, AAAI 2023*. AAAI.

Lin, S.; and Bercher, P. 2021. Change the World - How Hard Can that Be? On the Computational Complexity of Fixing Planning Models. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence, 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 *Proceedings of the 37th AAAI Conference on Artificial Intelligence AAAI 2023*. AAAI.

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

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *Proceedings of the 9th Knowledge Capture Conference, K-CAP 2017*, 1–8. ACM.

Olz, C.; and Bercher, P. 2019. Eliminating Redundant Actions in Partially Ordered Plans – A Complexity Analysis. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling, ICAPS 2019*, 310–319. AAAI.

Olz, C.; Biundo, S.; and Bercher, P. 2021. Revealing Hidden Preconditions and Effects of Compound HTN Planning Tasks - A Complexity Analysis. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, AAAI 2021*, 11903–11912. AAAI.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-Based Heuristics for Cost-Optimal Planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling, ICAPS 2014*, 226–234. AAAI.

Sreedharan, S.; Bercher, P.; and Kambhampati, S. 2022. On the Computational Complexity of Model Reconciliations. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI 2022*, 4657–4664. IJCAI.