Tight Bounds for Lifted HTN Plan Verification and Bounded Plan Existence

Pascal Lauer[1,2], **Songtuan Lin**[1,2], **Pascal Bercher**[1]

[1]School of Computing, The Australian National University, Canberra, Australia
[2]Saarland Informatics Campus, Saarland University, Saarbrücken, Germany
[1]firstname.lastname@anu.edu.au [2]lastname@cs.uni-saarland.de

Abstract

Plan verification is a canonical problem within any planning setting to ensure correctness. This problem is closely linked to the bounded plan existence problem. We analyze the complexity of these problems on lifted representations for Hierarchical Task Network (HTN) Planning. On top of the general analysis, we impose constraints on method orderings and the amount of tasks that methods decompose to. This pinpoints subclasses with lower complexity. Our results confirm the existence of more efficient algorithms when operating on the lifted, instead of grounded, representation.

1 Introduction

The most fundamental problem in planning is the *plan existence problem*, which asks whether a given planning problem has a solution (called a plan). In this work, we consider the closely related problem of *plan verification*, which asks whether a given action sequence is indeed a plan to a given planning problem. Once we can verify a plan, there is always an associated guess-and-check procedure. More importantly, plan verification is of huge practical importance. Without verification we had to trust the respective systems' implementation. While making use of an additional assurance should be "good practice" for any planning system developer, it becomes especially important for safety-critical applications, such as activity recognition while driving (Fernandez-Olivares and Perez 2020) or flying (Jamakatel et al. 2023). Or, when planning competitions are held, to not declare wrong/undeserved winners.

We focus on the complexity of plan verification in the context of Hierarchical Task Network (HTN) planning, a very expressive framework to describe planning problems (Erol, Hendler, and Nau 1996), that has gained significant attention in recent years (Bercher, Alford, and Höller 2019). Here, plan verification is also important, because some planners intrinsically rely on it as a subroutine. E.g. the TOAD planner (Höller 2021) overestimates the set of plans and then verifies if the computed action sequence was a plan.

In light of all mentioned use-cases, multiple approaches have been devised for plan verification. Examples include compilations to SAT (Behnke, Höller, and Biundo 2017; Lin, Behnke, and Bercher 2023), a compilation to HTN plan existence, (Höller et al. 2022), and adaptations of grammar parsing algorithms (Barták, Maillard, and Cardoso 2018; Barták et al. 2020; Ondrcková et al. 2023; Lin et al. 2023; Pantůčková and Barták 2023). Almost all approaches[1] operate on a grounded representation, where plan verification is known to be NP-complete in general and in PTIME if the entire model is totally ordered (Behnke, Höller, and Biundo 2015; Bercher, Lin, and Alford 2022; Lin and Bercher 2023). Although in practice problems are defined in a lifted representation like HDDL (Höller et al. 2020). We know that grounding a task can often become a bottleneck in large-scale tasks (Masoumi, Antoniazzi, and Soutchanski 2015; Wichlacz, Torralba, and Hoffmann 2019; Corrêa et al. 2020), which is why the standard verifier in classical (non-hierarchical) planning (Howey, Long, and Fox 2004) operates on the lifted representation. This highlights a significant gap between the theoretical framework and the practical approach most HTN verifiers take. It raises the question if and how complexity changes when using the lifted HTN representation directly.

Lin et al. (2024) initiated an investigation to answer this question, by analyzing the complexity of verification for grounded and lifted HTN planning problems. The analysis goes hand in hand with an analysis of the bounded plan existence problem, which is to determine whether there exists a plan that does not exceed a certain length bound. Their analysis distinguishes between unary and binary bounds, as discussed by Erol, Nau, and Subrahmanian (1991) and Bäckström and Jonsson (2011) in classical planning. The unary-bounded plan existence problem is particularly important, as it aligns with practical requirements: In real-world applications, a planner must output each action of the plan. To make up for the time this takes from the perspective of complexity theory, an appropriate solution is to provide a plan-length-sized parameter in the input.

However, Lin et al. (2024) only provided loose upper and lower bounds for the lifted case, and did not take important special cases into account, such as total order (which reduces the computational complexity in the ground setting (Behnke,

[1]The only exception is Barták et al. (2020). Though, empirical evidence suggests a benefit when adapting it to the grounded setting (Ondrcková et al. 2023).

Problem	Method Size	Computational Complexity	Hardness Proof	Membership Proof	Extended to TO- / UO- / PO-	Extended to TIHTN
Plan Verification	≥ 0	EXPTIME	Thm. 5	Prop. 1, Thm. 9	Cor. 11	Thm. 17
	≥ 1	PSPACE	Lin et al. (2024)	Prop. 1, Thm. 6	Cor. 8	Thm. 16
	$= 1$	PSPACE	Cor. 4	Prop. 1, Thm. 6	Cor. 8	Thm. 16
Unary-Bounded Plan Existence	≥ 0	EXPTIME	Prop. 1, Thm. 5	Thm. 9	Cor. 11	Thm. 17
	≥ 1	PSPACE	Lin et al. (2024)	Thm. 6	Cor. 8	Thm. 16
	$= 1$	PSPACE	Prop. 1, Cor. 4	Thm. 6	Cor. 8	Thm. 16
Binary-Bounded Plan Existence	≥ 0	NEXPTIME	Lin et al. (2024)	Thm. 12	Thm. 13	Cor. 21
	≥ 1	NEXPTIME	Thm. 13	Thm. 13	Thm. 13	Cor. 21
	$= 1$	PSPACE	Thm. 14	Thm. 14	Thm. 14	Cor. 22

Table 1: A summary of our complexity results. The Computational Complexity column lists completeness results. An extension means that tasks can be constrained to the according property without changing the complexity.

Höller, and Biundo 2015)). We fill this gap by providing tight bounds and taking such special cases into account. In addition to that, we even identified a new one of practical importance: Domains with non-empty methods, i.e. method size ≥ 1, for which verification drops in complexity. We note their practical relevance since an investigation of all IPC 2023 domains revealed that 11/24 total-order domains and 8/11 partial-order domains do not admit empty methods. In particular, we make the following contributions:

- For all loose bounds identified by Lin et al. (2024), we provide tight (matching) bounds.
- We provide tight bounds for the special cases of: Totally ordered and totally unordered methods (on top of the general case of partially ordered methods) and (non-)empty methods.
- We also investigate the complexity of classes for HTN problems with task insertion (TIHTN planning, (Geier and Bercher 2011; Alford, Bercher, and Aha 2015b)).

All results are provided for lifted plan verification and bounded plan existence, both for unary encoded bounds and binary encoded ones. Table 1 summarizes our findings.

2 Background

Classical Planning We start by defining (lifted) planning problems similar to Lauer et al. (2021) to mirror the STRIPS (Fikes and Nilsson 1971) part of PDDL (McDermott et al. 1998). We will then define HTN planning as an extension of the classical planning formalism. A lifted planning problem Π is a tuple $(\mathcal{P}, X, \mathcal{O}, \mathcal{A}, s_\mathcal{I}, \mathcal{G})$. $\mathcal{P}$ is the finite set of predicates. Each predicate $p \in \mathcal{P}$ is associated with a fixed arity $|p| \in \mathbb{N}$. X is the finite set of variables in the problem. $\mathcal{O}$ is the finite set of objects in the problem. We call the combination of $p(\vec{x})$ of a predicate symbol p with a sequence $\vec{x}$ of $|p|$ variables or objects (lifted) atom. The set of all lifted atoms is denoted by $\mathcal{P}^X$. An atom $p(\vec{x})$ is called grounded if $\vec{x}$ is a sequence of objects. The set of all grounded atoms is denoted by $\mathcal{P}^\mathcal{O}$. A state s is a set of grounded atoms. The set of all states is denoted by S. $s_\mathcal{I}$ is a state called initial state. The goal $\mathcal{G}$ is a set of grounded atoms. A (lifted) action a is a tuple $(sym(a), pre(a), add(a), del(a))$. $sym(a)$ is the unique action symbol. $pre(a)$, $add(a)$, and $del(a)$ are sets of lifted atoms. $\mathcal{A}$ is the set of actions in the problem.

An action is called grounded if all of its atoms are grounded. A planning problem is called grounded if all of its actions are grounded. $\mathcal{A}^\mathcal{O}$ is the set of all grounded actions that can be obtained by replacing the variables of the actions in $\mathcal{A}$. We ground a problem by replacing $\mathcal{A}$ with $\mathcal{A}^\mathcal{O}$. A grounded action $a \in \mathcal{A}^\mathcal{O}$ can be applied to a state $s \in S$ iff $pre(a) \subseteq s$. In this case we follow the notation of Hoffmann and Nebel (2001) to denote the (progressive) successor state as $\text{progr}(s,\ a) := (s \setminus del(a)) \cup add(a)$. $\text{progr}(s,\ a)$ is undefined if a can not be applied in s. The consecutive application of $a_1, \ldots, a_n \in \mathcal{A}$ to $s \in S$ and respective subsequent states is denoted by $\text{progr}(a_1, ..., a_n,\ s)$. A plan is an action sequence $a_1, \ldots, a_n$ that leads from $s_\mathcal{I}$ to a state containing the goal, i.e. $\text{progr}(a_1, \ldots, a_n,\ s_\mathcal{I}) \supseteq \mathcal{G}$. A planning problem is solvable iff there exists a plan. The amount of actions in the plan is called plan length.

HTN Planning HTN planning is an extension of classical planning that allows to restrict the solution (plan) space using a hierarchical structure. We define (lifted) HTN planning problems similar to Lin et al. (2024). A (lifted) HTN planning problem Π is a tuple $(\mathcal{P}, C, X, \mathcal{O}, \mathcal{A}, \mathcal{M}, s_\mathcal{I}, tn_\mathcal{I}, \mathcal{G})$. $\mathcal{P}, X, \mathcal{O}, \mathcal{A}, s_\mathcal{I}$ and $\mathcal{G}$ are defined as in classical planning. In the context of HTN planning we will also refer to actions as primitive tasks. The other type of tasks are compound tasks. Compound tasks are defined akin to atoms. C is the finite set of compound task symbols. Each $c \in C$ has a fixed associated arity $|c| \in \mathbb{N}$. A (lifted) compound task $c(\vec{x})$ is the combination of a compound task symbol c with a sequence $\vec{x}$ of $|c|$ variables or objects. The set of all (lifted) compound tasks is denoted as C^X. A compound task $c(\vec{x})$ is grounded if $\vec{x}$ consists only of objects. We indicate this, where relevant, by $c(\vec{o})$. The set of all grounded compound tasks is denoted $C^\mathcal{O}$. A (lifted) task network is a tuple $(T, \prec, \alpha)$ where T is a finite set of task identifiers, $\prec \subseteq T \times T$ a partial order over T, and $\alpha : T \to \mathcal{A} \cup C^X$ maps each identifier to a task. A task network is grounded iff all of its tasks, i.e. the tasks occurring in the image of α, are grounded. $tn_\mathcal{I}$ is a grounded task network called the initial task network. A method m is a tuple $(c(\vec{x}), tn)$, where $c(\vec{x}) \in C^X$ and tn is a task network. $\mathcal{M}$ denotes a finite set of methods.

A method is called grounded if both its head $c(\vec{x})$ and its task network tn are grounded. A planning problem is grounded if all of its methods and actions are grounded. Like with actions, $\mathcal{M}^{\mathcal{O}}$ denotes the set of all grounded methods that can be obtained by replacing the variables of the methods in $\mathcal{M}$. We ground an HTN planning problem by replacing $\mathcal{A}$ with $\mathcal{A}^{\mathcal{O}}$ and $\mathcal{M}$ with $\mathcal{M}^{\mathcal{O}}$.

A task network $(T, \prec, \alpha)$ is called primitive if all tasks $\alpha(T)$ are primitive. To find a plan, we first use methods to replace all compound tasks in a task network until it becomes primitive. Each step is called a decomposition. Similar to the action progression, we will formalize the decomposition only for grounded methods. Formally, a decomposition replaces a task id $t \in T$ that maps to a compound task $\alpha(t) = c(\vec{o})$ in a grounded task network $tn = (T, \prec, \alpha)$. $c(\vec{o})$ can be decomposed using a grounded method with matching head $m = (c(\vec{o}), (T_m, \prec_m, \alpha_m))$. In this case, we define the resulting task network, in a similar style as progr, as $\mathrm{decomp}(tn, t, m) := tn'$. In the the following we describe the construction of $tn' := (T', \prec', \alpha')$: The main idea is to remap the task identifiers T_m of the methods to "new" ones T_{new} that do not occur in the task identifiers T of the network to be decomposed, i.e. $T_{new} \cap T = \emptyset$. To do so we fix a T_{new} and a bijective function $\sigma_m : T_m \to T_{new}$. The replacement for T and α is straight-forward. Just remove the replaced and add the remapped part:

$$T' := T \setminus \{t\} \cup \sigma_m(T_m), \quad \alpha' := \alpha \setminus \{(t, c)\} \cup \sigma_m(\alpha_m)$$

For $\prec$ we identify the relations of the replaced task:

$$\prec_t := \{(t_1, t_2) \in \prec \mid t \in \{t_1, t_2\}\}$$

And the remapped new relations:

$$\begin{aligned}\prec_{t \leftrightarrow m} := &\{(\sigma_m(t_m), t') \mid t \prec t', t_m \in T_m\}\\ \cup &\{(t', \sigma_m(t_m)) \mid t' \prec t, t_m \in T_m\}\end{aligned}$$

Which allows for a similar definition as for T and α:

$$\prec' := \prec \setminus \prec_t \cup \sigma_m(\prec_m) \cup \prec_{t \leftrightarrow m}$$

A linearization of tasks in a task network is an ordering of tasks that agrees with its partial order. An action sequence $a_1, \ldots, a_n$ is called an action refinement iff it is a linearization of a primitive task network that is obtained as:

$$\mathrm{decomp}(\mathrm{decomp}(\mathrm{decomp}(tn_{\mathcal{I}}, t_1, m_1), ..., ...), t_k, m_k)$$

I.e. by repeated decomposition starting from the initial task network. The action sequence is a plan if it further fulfills the plan criterion of classical planning, i.e. $\mathrm{progr}(a_1, \ldots, a_n, s_{\mathcal{I}}) \supseteq \mathcal{G}$.

A commonly considered modification of HTN planning is to allow the insertion of additional actions into the plan. Formally, we modify the definition of a plan to so that a only sub-sequence of its actions needs to be an action refinement. We will refer to this extended problem as Task Insertion Hierarchical Task Network (TIHTN) planning.

Within both HTN and TIHTN planning we will analyze restrictions regarding the orderings in methods and how many tasks the method decomposes to. We start by defining the later. The size of a method $(t, (T, \prec, \alpha)) \in \mathcal{M}$ is $|T|$. For a given task, we will refer to method size $\geq c$ and $= c$ for $c \in \mathbb{N}$, if all methods have methods size $\geq c$, $= c$ respectively. Additionally, we will consider the following ordering constraints. A method $(t, (T, \prec, \alpha)) \in \mathcal{M}$ is called:

- *totally ordered* , if $\prec$ is a total order
- *unordered*, if $\prec = \emptyset$

A task is called totally ordered, or respectively unordered, if all of its methods are. The general case without further restriction is called *partially ordered*. We will prepend TO-/UO-, and PO where relevant, to indicate this. E.g. TO-HTN or TO-TIHTN planning for totally ordered.

We follow the definition by Lin et al. (2024), with minor adjustments, as one of our contributions is to tighten the open bounds from their work. As a result, we naturally build on their formalism. This formalism differs slightly from the original HTN formalism (Erol, Hendler, and Nau 1996). At least when considering only ordering constraints in task networks, which is common, our result extend to their formalism. We provide an elaborate discussion in section 8.

Analyzed Decision Problems For HTN planning (and its variations) we consider the following decision problems:

- *Plan Verification:* Given a planning problem Π and an action sequence π, is π a solution for Π?
- *Unary-Bounded Plan Existence:* Given a planning problem Π, $b \in \mathbb{N}_0^+$ in unary representation, is there a plan for Π of length $\leq b$?
- *Binary-Bounded Plan Existence:* Given a planning problem Π, $b \in \mathbb{N}_0^+$ in binary representation, is there a plan for Π of length $\leq b$?

Turing Machines In line with multiple preceding works on HTN planning (Alford, Bercher, and Aha 2015a; Alford et al. 2016b; Chen and Bercher 2022), multiple of our results are proven by a reduction from Turing machines. We define Turing machines similar to Rintanen (2004). We start by defining an Alternating Turing Machine (ATM) and restrict it to deterministic and non-deterministic Turing Machines as special cases. We assume w.l.o.g. that any Turing Machine controls one tape that is infinitely long only to the right-hand side. We restrict the tape alphabet, i.e. elements on the tape, to $\Gamma := \mathrm{Bits} \cup \{\sqcup\}$. $\sqcup$ is the blank symbol and $\mathrm{Bits} := \{0, 1\}$ the input alphabet. An ATM is a tuple $M = (Q, \triangleright, q_0, \text{\textiota})$. Q is a finite set of states. $\triangleright : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L, R\}}$ is the transition function, where L denotes a left head movement, and R represents a right movement. $q_0 \in Q$ is the initial state. $\text{\textiota} : Q \to \{\forall, \exists, \mathrm{accept}, \mathrm{reject}\}$ labels each state: $\forall$ for an alternating transition from the state, $\exists$ for a non-deterministic transition, accept for an accepting state and reject for a rejecting state. We assume that states $q \in Q$ with label $\text{\textiota}(q) \in \{\mathrm{accept}, \mathrm{reject}\}$ have no outgoing transitions, i.e. $\triangleright(q) = \emptyset$. And also that states $q \in Q$ with label $\text{\textiota}(q) \in \{\forall, \exists\}$ have outgoing transition(s), i.e. $\triangleright(q) \neq \emptyset$.

A configuration of M is a tuple $\Xi = (q, p, w) \in (Q \times \mathbb{N} \times \Gamma^*)$. q is the current state. $w \in \Gamma^*$ is the smallest word capturing all non-blank symbols and the head position. p is the head position within w. Transitions between configurations are performed according to $\triangleright$. The concrete construction is omitted due to space constraints. Ξ is accepting in $n \in \mathbb{N}$ steps if one of the following conditions holds:

- if $\text{\textiota}(q) = \mathrm{accept}$
- $\text{\textiota}(q) = \forall$ and all successors are accepting in $n - 1$ steps
- $\text{\textiota}(q) = \exists$ and one successor is accepting in $n - 1$ steps

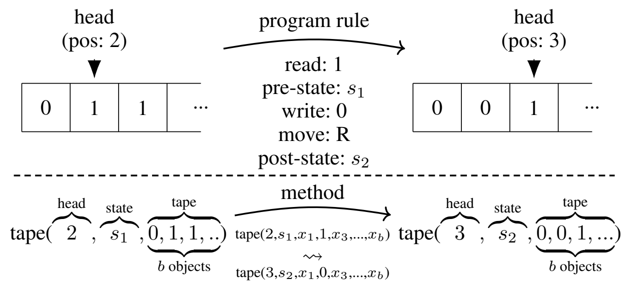
head
(pos: 2)
0 1 1 ...
program rule
read: 1
pre-state: s_1
write: 0
move: R
post-state: s_2
head
(pos: 3)
0 0 1 ...
method
tape(2, s_1, 0, 1, 1, ..)
head
state
tape
b objects
tape(2,s_1,x_1,1,x_3,...,x_b) $\rightsquigarrow$ tape(3,s_2,x_1,0,x_3,...,x_b)
tape(3, s_2, 0, 0, 1, ...)

Figure 1: Illustration of a task decomposition simulating a transition of a Turing machine with space bound $b \in \mathbb{N}$.

Ξ is rejecting in $n \in \mathbb{N}$ steps if one of the following conditions holds:

- $\iota(q) =$ reject
- $\iota(q) = \forall$ and one successor is rejecting in $n - 1$ steps
- $\iota(q) = \exists$ and all successors are rejecting in $n - 1$ steps

If Ξ is accepting or rejecting in n steps, we say it halts in n steps. Finally, a Turing machine is accepting/rejecting/halting in n steps for input w_I iff configuration $(q_0, 0, w_I)$ is.

A nondetermistic Turing Machine (NTM) is an ATM with the restriction that there is no $q \in Q$ so that $\iota(q) = \forall$. A (deterministic) Turing Machine (DTM) is an NTM with the restriction that for all $q \in Q$ and $w \in \Gamma$ it holds $|\rhd(q, w)| = 1$.

3 The Relation of HTN Plan Verification and Unary-Bounded Plan Existence

We start by relating plan verification and unary-bounded plan existence. This simplifies proofs in the following section. Lin et al. (2024) showed that the complexity of verification and unary-bounded plan existence in grounded HTN planning are the same. This sparks question if there is a direct relation of both problems that generalizes over certain constraints. Höller et al. (2022) partially answered the question by presenting an encoding of verification via plan existence. Though, the construction is on grounded HTN planning tasks and relies on a modification of concrete grounded actions. Thus there is no canonical extension to the lifted encoding. In the following we present an alternative construction for lifted HTN planning.

Proposition 1. *Plan verification for lifted HTN planning tasks is polynomial-time reducible to the Unary-bounded plan existence for lifted HTN planning tasks.*

Proof. For action sequence $\pi = a_1(\vec{o}_1), ..., a_n(\vec{o}_n)$, HTN planning problem $\Pi = (\mathcal{P}, C, X, \mathcal{O}, \mathcal{A}, \mathcal{M}, s_\mathcal{I}, tn_\mathcal{I}, \mathcal{G})$, we will modify Π to allow only plans that resemble π. To do so we introduce a predicate $used\_pred$ of arity $\max_{a \in \mathcal{A}} |a| + 2$ which will mark the actions used in the task. To track each application action $a \in \mathcal{A}$, we define an atom $used_a$ as:

$$used\_pred(x_{step}, sym(a), inst_1, ..., inst_{|used\_pred|-2})$$

Where $inst_1, ..., inst_{|X(a)|}$ are the $|X(a)|$ pair-wise distinct variables $X(a)$ of a. And $inst_{|X(a)|+1}, ..., inst_{|used\_pred|-2}$ is the new object $\bot \notin \mathcal{O}$.

To map $x_{step} \notin X(a)$ to the number corresponding to the amounts of actions applied already, we use a predicate $count$ to track the current action count and predicate $next$ to identify the next higher counter. The final construction is:

$$\mathcal{P}' = \mathcal{P} \cup \{count, next, used\_pred\}$$
$$\quad \textit{with } |count| = 1, |next| = 2, |used\_pred| = \max_{a \in \mathcal{A}} |a| + 2$$
$$\mathcal{O}' = \mathcal{O} \cup \{0, ..., n-2\} \cup \{\bot\}$$
$$s_\mathcal{I}' = s_\mathcal{I} \cup \{next(i, i+1) \mid i \in \{0, ..., n-2\}\}$$
$$\quad \cup \{count(0)\}$$
$$X' = \bigcup_{a \in \mathcal{A}} X(used_a) \cup X \cup \{x_{step}, x_{step'}\}$$
$$\mathcal{A}' = \{(sym(a), pre(a) \cup \{count(x_{step}), next(x_{step}, x_{step'})\},$$
$$\quad add(a) \cup \{used_a, count(x_{step'})\},$$
$$\quad del(a) \cup \{count(x_{step})\}, c(a)) \mid a \in \mathcal{A}\}$$

Here we can apply at most n actions and each application of an action $a(\vec{o})$ in step $i \in \{1, ..., n\}$ will add the atom $used\_pred(i, sym(a), \vec{o})$ to the state. We denote this atom by $used_{i,a(\vec{o})}$ and modify the goal to require action applications according to π. I.e., we define the set $\mathcal{G}_{used}$ as:

$$\{used_{1,a_1(\vec{o}_1)}, ..., used_{n,a_n(\vec{o}_n)}\}$$

And output an HTN planning problem $\Pi' = (\mathcal{P}', C, X', \mathcal{O}', \mathcal{A}', \mathcal{M}, s_\mathcal{I}', tn_\mathcal{I}, \mathcal{G} \cup \mathcal{G}_{used})$ and unary plan length n. Here a plan exists iff π was a plan in Π. $\square$

Note that the task produced as outcome of the reduction preserves all analyzed restrictions w.r.t. the input task.

Corollary 2. *The reduction provided in Proposition 1 preserves the following constraints (if applicable): unordered, totally ordered, methods size = 1 and method size ≥ 1.*

4 HTN Plan Verification

Here, we analyze the complexity of HTN plan verification for lifted HTN planning tasks under different restrictions. As per Section 3 we will here only present hardness proofs and tighten the bound via membership proofs for the unary-bounded plan existence problem in Section 5. All proofs in this Section reduce from differently constrained Turing machines. To understand the main intuition, take a look at Figure 1. It illustrates how to simulate a transition of a space-bounded Turing machine using a single task decomposition. In particular, it illustrates a transition for a program rule where the head moves to the right after reading 1 and writing 0. We encode the tape as a compound task with arity $b + 2$. The first position is used to annotate the head position, the second is the state. The remaining b objects resemble the relevant part of the tape. The Turing machine rule writes a 0 after reading 1 and then moves to the right. The decomposition rule is encoded for head position 2 and enforces 1 to be at this position. It then changes the task to have a 1 at this position and changes the Turing state accordingly. Note that this method only depends on the value of the head position, the head position itself and the state. As we generate one method per tape position and transition rule, this allows to generate an at most quadratic amount of methods per Turing rule (cubic w.r.t. the input).

We start by providing a reduction of deterministic PSPACE Turing machines. This also serves as alternate proof of Theorem 1 from Lin et al. (2024). More importantly, it will canonically extend to a reduction from alternating polynomial space restricted Turing machines in Theorem 5, which will prove hardness for APSPACE = EXPTIME (Chandra, Kozen, and Stockmeyer 1981).

Theorem 3. *Plan verification for lifted HTN planning tasks is PSPACE-hard.*

Proof. We reduce from the acceptance problem for a deterministic Turing Machine that is space-bounded w.r.t. a polynomial p. This is: Given a Turing machine $M = (Q, \triangleright, q_0, \iota)$, input w, and two bounding parameters $1^c, 1^n$, does M accept w within space $b := p(n) + c$?

As in Figure 1 we will model the tape using a compound task with id $tape$ and arity $b + 2$ where: The first argument to the compound task denotes the head position of the Turing machine. The second denotes the current Turing Machine state. The remaining arguments capture the tape content. (b elements starting from the leftmost position.) So, set $C = \{\langle tape, (x_{head}, x_{state}, x_{t_1}, ..., x_{t_b})\rangle\}$. We extend input w, by adding $\sqcup$, or cut it, by omitting the end, to length b. The result is denoted by w^*. This allows to create the initial task as $t_{\mathcal{I}} = tape(0, q_0, w_1^*, ..., w_b^*)$. With $t_{\mathcal{I}}$ we associate the initial task network $tn_{\mathcal{I}}$ that only contains $t_{\mathcal{I}}$. This means the objects in our task are $\mathcal{O} = \{0, 1, ..., b\} \cup Q$ for the bits and head position and the Turing states respectively. We will not need any predicates, i.e. $\mathcal{P} = \emptyset, s_{\mathcal{I}} = \emptyset$. And set the primitive tasks $\mathcal{A} = \{a\}$ to one empty action $a := \langle 0, \{\}, \{\}, \{\}\rangle$.

We will now encode tape transitions using task decompositions. For convenience we will first define tasks $t_{w,i,q}$ that are $tape$ compound tasks with fixed state $q \in Q$, head position $i \in \{1, ..., b\}$ and bit $w \in \{0, 1\}$ at tape position i. All other arguments are distinct variables. Formally we define:

$$t_{w,i,q} = \begin{cases} tape(1, q, w, x_2, ..., x_b) & \text{, if } i = 1 \\ tape(b, q, x_1, ..., x_{b-1}, w) & \text{, if } i = b \\ tape(i, q, x_{x_1}, ..., x_{i-1}, w, x_{i+1}, ..., x_b) & \text{, o/w} \end{cases}$$

Now we will encode the decomposition for each transition rule $r = \{(q_1, w_1) \mapsto \{(q_2, w_2, lr)\}\} \in \triangleright$ and each tape position $i \in \{1, ..., b\}$ (except for $(lr, i) \in \{(L, 1), (R, b)\}$). This is that we encode a method $m_{r,i} := \langle t_{w_1,i,q_1}, (\{1\}, \emptyset, \{1 \mapsto t\})\rangle$ that decomposes into the one new task t, defined as:

$$t = \begin{cases} a & \text{, if } \iota(q_2) = \text{accept} \\ t_{w_2,i+1,q_2} & \text{, if } lr = R \\ t_{w_2,i-1,q_2} & \text{, if } lr = L \end{cases}$$

$\mathcal{M}$ denotes the collection of all such methods. X is the collection of all variables occurring in $\mathcal{M}$. We output $(\mathcal{P}, C, X, \mathcal{O}, \mathcal{A}, \mathcal{M}, s_{\mathcal{I}}, tn_{\mathcal{I}}, \mathcal{G})$. and plan $a()$. Our construction allows mirroring any configuration of a b-space-bounded Turing machine as a composite task. Each decomposition directly corresponds to the mirrored configuration's transitions of the Turing machine. A task can be decomposed into a iff the according configuration would be accepting. Thus the action sequence a is a plan iff the Turing machine is accepting. □

In the planning task constructed in the proof, each compound task is decomposed into exactly one task. This allows us to conclude:

Corollary 4. *Plan verification for lifted HTN planning tasks with method size = 1 is PSPACE-hard.*

We will now extend the proof to alternating Turing machines. The alternation will be covered by methods producing multiple compound tasks that resemble all alternating outcomes. To be able to map a correct decomposition to a plan, we will decompose to the empty task network if the compound task transitions to an accepting state. This allows to verify the empty plan as valid plan.

Theorem 5. *Plan verification for lifted HTN planning tasks is EXPTIME-hard.*

Proof. We extend the proof of Theorem 3 to Alternating Turing Machines. We assume the same context as in Theorem 3 except that M is an Alternating Turing machine. We will now also ignore actions, i.e. $\mathcal{A}' = \emptyset$. The only other thing we change about the reduction are the methods $\mathcal{M}$. Before we adjust the method definition, we want to extend the task shorthand notation to $t_{q_1,w_1,q_2,w_2,lr}$ for fixed states $q_1, q_2 \in Q$, the head position $i \in \{1, ..., b\}$ and bits $w_1, w_2 \in \{0, 1\}$ as:

$$t_{q_1,w_1,q_2,w_2,lr} = \begin{cases} \bot & \text{, if } \iota(q_2) = \text{accept} \\ t_{w_2,i+1,q_2} & \text{, if } lr = R \\ t_{w_2,i-1,q_2} & \text{, if } lr = L \end{cases}$$

Now for each transition rule $r = \{(q_1, w_1) \mapsto \{(q_2, w_2, lr_2), ..., (q_n, w_n, lr_n)\}\} \in \triangleright$ we build the following methods depending on the transition label:

1. If $\iota(q_1) = \exists$ we define for $k \in \{2, ..., n\}, i \in \{1, ..., b\}$ a method $m_{r,k,i}$ that decomposes the tasks t_{w_1,i,q_1} into T_k, defined as:

$$T_k = \{t_{q_1,w_1,q_k,w_k,lr_k}\} \setminus \{\bot\}$$

2. If $\iota(q_1) = \forall$ we define for $i \in \{1, ..., b\}$ a method $m_{r,i}$ that decomposes the tasks t_{w_1,i,q_1} into T_k, defined as:

$$T_k = \{t_{q_1,w_1,q_2,w_2,lr_2}, ..., t_{q_1,w_1,q_n,w_n,lr_n}\} \setminus \{\bot\}$$

$\mathcal{M}'$ denotes the collection of all such methods. We replace the primitive tasks by $\mathcal{A}'$ and methods by $\mathcal{M}'$ in the task from Theorem 3 to obtain Π'. We output Π' and the empty plan. As in Theorem 3 we ensure that any decomposition mirrors the transitions in the Turing machine vice versa. The decomposition to multiple tasks ensures that all tasks need to be decomposed to the empty tasks which is the case if and only if all corresponding configurations lead to an accepting state. As APSPACE = EXPTIME (Chandra, Kozen, and Stockmeyer 1981) the reduction proves the claim. □

Algorithm 1: NTM Verifier Unary-Bounded Plan Existence

```
1: **Input**: Unary bound $b$, an HTN problem $\Pi = (\mathcal{P}, C, X, \mathcal{O}, \mathcal{A}, \mathcal{M}, s_\mathcal{I}, tn_\mathcal{I}, \mathcal{G})$ with method size $\geq 1$
2: $r_b \leftarrow |C^\mathcal{O}| + 1, \quad r_c \leftarrow 0$
3: $tn \leftarrow tn_\mathcal{I}$
4: while $tn$ is not primitive do
5:     guess compound task $t$ from $tn$, $m \in \mathcal{M}^\mathcal{O}$
6:     if $m$ can not decompose $t$ then
7:         reject
8:     end if
9:     $r_c \leftarrow r_c + 1$ if method size of $m$ is 1 else 0
10:    if $r_c > r_b$ then
11:        reject
12:    end if
13:    $tn \leftarrow \mathrm{decomp}(tn, t, m)$
14:    if $tn$ has more than $b$ tasks then
15:        reject
16:    end if
17: end while
18: guess linearization $a_1, \ldots, a_n$ of $tn$
19: accept if $a_1, \ldots, a_n$ is a plan, otherwise reject
```

5 Unary-Bounded HTN Plan Existence

Here we provide the membership proofs for the lifted unary-bounded plan existence problem. This will tighten the bounds for both this problem and lifted plan verification. We will start with the special cases when there exists no method that decomposes to an empty task network.

Theorem 6. *Unary-bounded plan existence for lifted HTN planning tasks with method size ≥ 1 is in PSPACE.*

Proof. We implement a non-deterministic guess-and-check algorithm for unary-bounded plans as presented in Algorithm 1. The condition in line 14 ensures that no more than b elements are added to the task network tn. This ensures that any considered action sequence in line 18 has size at most b. It does not reject any plan, as the task network size is a lower bound on sequence length (which in this case would exceed b).

Now, if we would ignore line 10, we would already have an algorithm that returns a solution of size b if one exists as it non-deterministically considers all possibilities for action sequences of size at most b (that can be obtained via decomposition and obey ordering restrictions). The condition of line 10 makes sure that the recursion count r_c exceeds the recursion bound r_b. (If this doesn't happen then the algorithm is terminated after the check in line 14 as the task network grows.) This ensures that the algorithms encounters no endless loop when always decomposing to exactly one compound task. If the condition were to reject a plan, an earlier guess can always lead to a valid plan, as the amount of decompositions to one tasks exceeds the amount of grounded compounds tasks. Thus the algorithm always terminates and accepts/rejects correctly.

It remains to show that the algorithm takes only polynomial space. We can bound the binary representation $|C^\mathcal{O}| \leq (|C| + |\mathcal{O}|)^{\max_{c \in C} |c| + 1}$ which allows to bound r_b's binary representation with $O(\log((|C| + |\mathcal{O}|)^{\max_{c \in C} |c|})) \subseteq O(\log((|C| + |\mathcal{O}|)) \cdot \max_{c \in C} |c|)$. This is polynomially bounded w.r.t. Π. Guessing grounded method $m \in \mathcal{M}^\mathcal{O}$ is possible in linear space and time by first guessing method schema, then method replacements. As tn never exceeds b it can be stored in polynomial space. Thus the algorithm is in NPSPACE = PSPACE (Savitch 1970). □

Note that method size $= 1$ is a more constraining case of ≥ 1. Thus the combination of Cor. 2, Cor. 4 and Thm. 6 allows to obtain the following complexity result:

Corollary 7. *Plan verification and unary-bounded plan existence for lifted HTN planning tasks with methods size $= 1$ or ≥ 1 is PSPACE-complete.*

When considering ordering constraints, complexity can only decrease because any proof of membership for the general case also applies to the restricted case. The hardness proof in Thm. 3 has one task in the initial task network and method size $= 1$. Thus it is unordered and totally ordered, which allows to conclude:

Corollary 8. *Cor. 7 extends to TO- & UO-HTN planning.*

The grounded representation can be exponentially larger than the lifted input. This makes its creation generally more costly than verifying and finding bounded plans directly on the lifted representation for this subclass. As there is no HTN plan verification that does not ground except for Barták et al. (2020), which takes exponential time (and does not use informed heuristics yet). Our novel PSPACE result now gives hope for better approaches.

E.g. by hinting an encoding into grounded classical planning, where plan existence is PSPACE-complete (Erol, Nau, and Subrahmanian 1991). Utilizing this connection in different contexts of HTN planning, has proven to be very promising before (Höller et al. 2018, 2019). Thus it is reasonable to believe that this hints yet another way to benefit from the various successful approaches (Seipp and Helmert 2013; Fišer, Torralba, and Hoffmann 2022; Corrêa and Seipp 2022) developed in classical planning.

We will now continue to analyze the general case, where a task may be decomposed to the empty task network.

Theorem 9. *Unary-bounded plan existence for lifted HTN planning tasks is in EXPTIME.*

Proof. We adjust Algorithm 1. We change the check in line 9 to also increase on method size 0. We extend the code before line 13 with a guess of some tasks that we remove from m's decomposition result. (m is treated as a locally created object. Modifying it does not affect the original method structure.) We check via an oracle if the selected tasks for deletion can be deleted, if not, we reject. By allowing this deletion of tasks, we extend the correctness argument from Theorem 6 to the general case. In particular, tasks that would be removed through decomposition can now be deleted by this guess. Consequently, assuming a non-deterministic guess that deletes all such tasks establishes the number of tasks in tn as a lower bound on the number of actions in the plan. As in Theorem 6 we yield a PSPACE algorithm (under assumption of a constant cost oracle). It follows that there is also an EXPTIME algorithm (under assumption of a constant cost oracle). We will now argue that we can replace the oracle with an EXPTIME algorithm. Calling an EXPTIME algorithm exponentially times remains EXPTIME. So, the final construction yields an EXPTIME verifier (even if methods can decompose to the empty task).

To construct an EXPTIME algorithm for this oracle we utilize a result from Behnke, Höller, and Biundo (2015). They show that checking if a task can be decomposed to the empty task network is in P for grounded HTN planning tasks[2]. Thus, grounding the task (in exponential time) and using the polynomial time check on the grounded task yields the EXPTIME algorithm. □

As we have now proven the lower bound for unary-bounded plan verification and upper bound for plan verification, it allows us to conclude by Prop. 1:

Corollary 10. *Plan verification and unary-bounded plan existence for lifted HTN planning tasks is EXPTIME-complete.*

Again, to extend the result to consider ordering constraints, we only need to analyze the hardness proof. In the general case, again any ordering is completely irrelevant, as the plans are empty.

Corollary 11. *Cor. 10 extends to TO- & UO-HTN planning.*

Verification and finding a bounded plan, on the potentially exponentially large grounded representation, is NP-complete (Lin et al. 2024). Assuming that P $\neq$ NP, we can now observe that it is generally easier to use the lifted representation directly, as the grounder output may be exponential. Additionally, this hints another encoding into classical planning. In particular, into delete-free lifted classical planning, where the plan existence is known to be EXPTIME-complete (Erol, Nau, and Subrahmanian 1991). This particular fragment of classical planning has received a lot of attention and progress lately (Corrêa et al. 2021, 2022; Lauer et al. 2025). Thus it could serve as yet another performant way for verification in practice.

6 Binary-Bounded Plan Existence

We continue to analyze the complexity of plan existence with a binary bound. Lin et al. (2024) have only proven this problem to be NEXPTIME-hard. Here, we will show it is NEXPTIME-complete. Further, we will extend the analysis to our analyzed constraints. We will see that here complexity does not change for method size ≥ 1. We start by analyzing the general case without restrictions.

Theorem 12. *Binary-bounded plan existence for lifted HTN planning tasks is NEXPTIME-complete.*

Proof. Hardness is proven by Lin et al. (2024). For membership we use the following NEXPTIME verifier: Given length bound k, guess an action sequence of length at most k. Ground the task (in exponential time). We pass the (exponentially-sized) action sequence with the (exponentially-sized) grounded task to the NP verifier of Behnke, Höller, and Biundo (2015) from the grounded setting. This remains NEXPTIME. By considering every plan of length k the verification is trivially correct. □

This allows us to determine the same complexity for the case when restricting to methods size ≥ 1.

Theorem 13. *Binary-bounded plan existence for lifted HTN planning tasks is NEXPTIME-complete. This result extends to method size ≥ 1 in TO-, UO- and PO-HTN planning.*

Proof. Membership is proven by Thm. 12, as we consider a more constrained case. For hardness we reduce from the binary-bounded plan existence problem for lifted classical planning, which is known to be NEXPTIME-complete (Erol, Nau, and Subrahmanian 1991). The bound remains the same. To simulate a classical planning task with HTN planning we use the construction from Erol, Hendler, and Nau (1996, Sec. 3.3). This is to have one compound task that can be decomposed into an arbitrary action or twice this compound task. On this construction one can impose arbitrary ordering constraints, without restricting the action refinements, as any arbitrary sequence remains to be a refinement. Further it never decomposes to the empty task network, which proves the claim. □

Lastly there is a difference for methods size $= 1$, which we can observe from combinations of our previous results.

Theorem 14. *Binary-bounded plan existence for lifted HTN planning tasks with method size $= 1$ is PSPACE-complete. This result extends to method TO- & UO- HTN planning.*

Proof. Hardness follows from a reduction from the problem with a unary bound. We output the same task with the bound converted to binary.

To observe membership we use Alg. 1 (with a binary bound instead of unary). As we never decompose to more than 1 task, tn will not increase in size and the algorithm uses only polynomial space. □

7 TIHTN Planning

Here we extend the previous result to TIHTN planning. We will start with the results for plan verification and unary-bounded plan existence. We provide the same hardness relation as in Prop. 1, which can just be observed by the same construction.

Corollary 15. *Prop. 1 and Cor. 2 extend to TIHTN planning.*

[2]Def. 1 of Behnke, Höller, and Biundo (2015) shows how to identify the tasks that can be decomposed to the empty task network in $\leq k$ steps in time polynomial time w.r.t. the planning problem and k. Lem. 1 of their work then concludes that there is a choice for k, that is polynomially bounded by the size of the planning problem, so that the identified tasks for all $k + i$ for $i > 0$ are the same. This concludes that detecting whether a task is decomposable to the empty task network is in PTIME.

This allows us to provide a hardness lower-bound for plan verification and membership upper-bound like in Sections 3, 4 and 5, which we will closely adhere to.

Theorem 16. *Plan verification and unary-bounded plan existence for lifted TIHTN planning tasks with method size $= 1$ or ≥ 1 is PSPACE-complete. The result extends TO-, UO- and PO-TIHTN planning.*

Proof. For hardness we can use exactly the same argumentation as in the case without task insertion (Thm. 3, Cor. 4).

For membership adjust the Verification procedure presented in Alg. 1 to non-deterministically guess whether to add a (guessed) task or continue decomposing. □

Theorem 17. *Plan verification and unary-bounded plan existence for lifted TIHTN planning tasks is EXPTIME-complete. This extends TO-, UO- and PO-TIHTN planning.*

Proof. Like in Thm. 16, for hardness we can just recreate the proof for normal HTN planning from Thm. 5.

And for membership adjust Alg. 1 to non-deterministically guess whether to add a (guessed) task or continue decomposing. □

For binary-bounded plan existence we will show the hardness for the more constrained case with method size ≥ 1 and membership for the general case. We start with hardness, for which we can use exactly the same argument as in Thm. 13 where we generate arbitrary action sequences to simulate classical planning. Allowing task insertion will not change the set of plans, as any action sequence is a refinement.

Corollary 18. *Binary-bounded plan existence for lifted TIHTN planning tasks with method size $= 1$ is NEXPTIME-hard.*

We first show NP membership for the grounded case to then conclude with a similar argument as in Thm. 12.

Proposition 19. *Plan verification for grounded TIHTN planning tasks is in NP.*

Proof. As r_b in Alg. 1 is an upper bound on the amount of tasks in the problem, in the grounded case, one can simply adjust it to be the number of grounded tasks in the problem. Then the verification algorithm is in NP. We can use the adjustments from Thm. 17 to extend it to TIHTN planning. Thus unary-bounded plan existence for grounded TIHTN is in NP. And so by Cor. 15 also plan verification. □

Now we can basically reproduce the proof of Thm. 12 for TIHTN planning.

Theorem 20. *Binary-bounded plan existence for lifted TIHTN planning tasks is in NEXPTIME.*

Proof. The membership check is: Guess a plan of size at most k and uses the verifier from Prop. 19 to accept or reject. The correctness argument is the same as in Thm. 12. □

This concludes the final statement as we sandwich the result. Note that like before we can impose ordering constraints in the argumentation before without changing the result.

Corollary 21. *Binary-bounded plan existence for lifted TIHTN planning tasks is NEXPTIME-complete. This extends to problem with method size to ≥ 0 and ≥ 1 in TO-, UO- and PO-TIHTN planning.*

Thus it remains to show that we can extend this to method size ≥ 1 Here we can mirror the argumentation for the case without task insertion.

Theorem 22. *Binary-bounded plan existence for lifted TIHTN planning tasks with method size $= 1$ is PSPACE-complete. The result extends TO-, UO- and PO-TIHTN planning.*

Proof. Hardness is proven by the unary case in in Thm. 16. For membership we follow the same argument as for Thm. 14 holds. I.e. one can use Alg. 1 with a binary bound. □

This observation concludes that complexity does not change for TIHTN planning. However, it is unclear if the lifted representation offers an advantage over the grounded one in TIHTN planning. Therefore we provide a classification for the primary problems of interest in the grounded setting

Theorem 23. *Plan verification and unary-bounded plan existence for grounded TIHTN planning tasks is NP-complete.*

Proof. The membership upper-bound was proven in Prop. 19. The hardness lower-bound is can be created by reusing the construction from Thm. 3 in Lin and Bercher (2023). In the constructed HTN problem it holds for all $a \in \mathcal{A}$ that $pre(a) = add(a) = del(a) = \emptyset$. Here the hardness lies in the decomposition, as any action can be inserted without change. Thus, the result extends to TIHTN planning. □

The absence of a change in complexity highlights the challenge task decomposition poses. This suggests that, if we do not want to impose limits on the decomposition, complexity is unlikely to drop further. Transforming these problems into classical planning could be a viable approach, as it at least equips us with good performing solvers. Especially within the PSPACE fragment, where it becomes possible to use a plethora of state-of-the-art planners. Prior research has successfully utilized bounded classical planning encodings to find Hierarchical Task Network (HTN) plans (Alford, Kuter, and Nau 2009; Alford et al. 2016a; Behnke et al. 2022). Though their bounds relate to the decomposition directly. Our results suggest that *just* bounding plan length can be a promising alternative.

8 Relation to Other HTN Formalisms

To the best of our knowledge, the decision problems we study in this paper were not studied for any HTN formalism, including original HTN formalism (Erol, Hendler, and Nau 1996). To draw conclusions about the complexity in the original formalism, from our results, we point out the three aspects in which our formalism deviates:

1. Our solution definition differs in being an action sequence, rather than a (partially ordered) task network that admits such a sequence.

2. In our formalism, the initial task network is always grounded, whereas Erol, Hendler, and Nau define it to be lifted.
3. In our task networks, we consider only ordering constraints, whereas Erol, Hendler, and Nau additionally consider arbitrary first-order logic constraints.

We start by arguing that (1.) and (2.) do not affect the results. For (1.), solutions can be easily converted, as the complexity classes are at least NP and thus can be resolved by guessing the corresponding sequence, or testing all combinations for EXPTIME. Thus, the results are affected by this distinction. We opted for a linearized sequence, as this matches the format used for plans in practice. This makes it closer to reality, which makes the most sense in the context of plan verification.

For (2), one can use a compilation that introduces a grounded compound task in the initial task network and a method that decomposes into the original lifted task network. Note that this may affect method size. Though, the changed method size only affects Alg. 1 and its variations. It can be adjusted by adding one guessing step to ground the initial task network. Thus, the results are affected by this distinction. We opted for enforcing a grounded initial task network because it simplifies the definitions significantly in our context.

We strongly believe that the results extend to (3) as well. The complexity classes are high enough to allow the additional constraints to be resolved through guessing in the membership proofs, although we have not formally studied this. For hardness, the results definitely extend, as the deviation in (3) simply increases expressivity. We preferred to analyze the problems without including (3), as this simplification is found in most works surrounding HTN planning. Moreover, there are no benchmarks that use the constraints unique to Erol, Hendler, and Nau's formalism, so we found this decision to be a natural choice.

9 Conclusion

In this work, we investigated the computational complexity of the plan verification problem (is a given action sequence a solution to a given planning problem?) and bounded plan existence problem (is there an action sequence within a certain length bound that is a solution to a given planning problem?) in HTN planning. More specifically, we investigated the complexity of the practically motivated scenario where a grounded plan (as generated by any planning system) should be verified, or respectively generated, but a *lifted* model is provided as its input, specified by input languages such as HDDL. Thus, our analysis fills an important gap in the HTN theory, as most verification results were previously based on a fully ground model. On top of providing tight complexity bounds for this lifted setting, we also identified an important novel special case, problems with only non-empty decomposition methods, for which we were able to provide lower bounds (complexity drops from EXPTIME-complete to just PSPACE-complete). We furthermore provide tight complexity bounds when task insertion is allowed as in TIHTN planning. Lastly, all investigations are done for partially ordered (the general case), totally ordered, and also totally unordered task networks (i.e. without ordering constraints). Interestingly, while it is known from the literature that total order drops the complexity from NP-complete to polynomial-time for grounded problems, none of those three cases has an influence of the computational complexity when the input model is lifted. As a last (and minor) contribution, we also point out the relationship between lifted plan verification and bounded plan existence. Our results strongly suggest that working directly on a lifted model can pay-off. The results guarantee that lifted approaches at least prevail when grounding becomes intractable, which was observed in multiple domains recently. This paves the way for future research avenues, to identify such lifted approaches in an empirical setting. We point at a promising avenue as the use of classical planning by linking decision problems of the same complexity.

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – ProjectID 232722074 – SFB 1102. Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; and Aha, D. 2016a. Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *ICAPS*.

Alford, R.; Bercher, P.; and Aha, D. 2015a. Tight bounds for HTN planning. In *ICAPS*.

Alford, R.; Bercher, P.; and Aha, D. W. 2015b. Tight Bounds for HTN Planning with Task Insertion. In *IJCAI*.

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way. In *IJCAI*.

Alford, R.; Shivashankar, V.; Roberts, M.; Frank, J.; and Aha, D. W. 2016b. Hierarchical Planning: Relating Task and Goal Decomposition with Task Sharing. In *IJCAI*.

Barták, R.; Maillard, A.; and Cardoso, R. C. 2018. Validation of Hierarchical Plans via Parsing of Attribute Grammars. In *ICAPS*.

Barták, R.; Ondrčková, S.; Maillard, A.; Behnke, G.; and Bercher, P. 2020. A novel parsing-based approach for verification of hierarchical plans. In *ICTAI*.

Behnke, G.; Höller, D.; and Biundo, S. 2017. This is a solution!(... but is it though?)-verifying solutions of hierarchical planning problems. In *ICAPS*.

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

Behnke, G.; Pollitt, F.; Höller, D.; Bercher, P.; and Alford, R. 2022. Making translations to classical planning competitive with other HTN planners. In *AAAI*.

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

Bercher, P.; Lin, S.; and Alford, R. 2022. Tight Bounds for Hybrid Planning. In *IJCAI*.

Bäckström, C.; and Jonsson, P. 2011. All PSPACE-complete planning problems are equal but some are more equal than others. In *SOCS*.

Chandra, A. K.; Kozen, D. C.; and Stockmeyer, L. J. 1981. Alternation. *Journal of the ACM*, 28.

Chen, D. Z.; and Bercher, P. 2022. Flexible FOND HTN Planning: A Complexity Analysis. In *ICAPS*.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-relaxation heuristics for lifted classical planning. In *ICAPS*.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Frances, G. 2020. Lifted successor generation using query optimization techniques. In *ICAPS*.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Frances, G. 2022. The FF heuristic for lifted classical planning. In *ICAPS*.

Corrêa, A. B.; and Seipp, J. 2022. Best-First Width Search for Lifted Classical Planning. In *ICAPS*.

Erol, K.; Hendler, J.; and Nau, D. S. 1996. Complexity results for HTN planning. *Ann. Math. Artif.*, 18.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1991. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. Technical Report CS-TR-2797, UMIACS-TR-91-154, UMD.

Fernandez-Olivares, J.; and Perez, R. 2020. Driver Activity Recognition by Means of Temporal HTN Planning. In *ICAPS*.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *AIJ*, 2.

Fišer, D.; Torralba, A.; and Hoffmann, J. 2022. Operator-Potential Heuristics for Symbolic Search. In *AAAI*.

Geier, T.; and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *IJCAI*.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *JAIR*, 14.

Höller, D. 2021. Translating totally ordered HTN planning problems to classical planning problems using regular approximation of context-free languages. In *ICAPS*.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *AAAI*.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2018. A generic method to guide HTN progression search with classical heuristics. In *ICAPS*.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2019. On Guiding Search in HTN Planning with Classical Planning Heuristics. In *IJCAI*.

Höller, D.; Wichlacz, J.; Bercher, P.; and Behnke, G. 2022. Compiling HTN plan verification problems into HTN planning problems. In *ICAPS*.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *ICTAI*.

Jamakatel, P.; Bercher, P.; Schulte, A.; and Kiam, J. J. 2023. Towards Intelligent Companion Systems in General Aviation using Hierarchical Plan and Goal Recognition. In *HAI*.

Lauer, P.; Torralba, Á.; Fiser, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *IJCAI*.

Lauer, P.; Torralba, Á.; Höller, D.; and Hoffmann, J. 2025. Continuing the Quest for Polynomial Time Heuristics in PDDL Input Size: Tractable Cases for Lifted. In *ICAPS*.

Lin, S.; Behnke, G.; and Bercher, P. 2023. Accelerating SAT-Based HTN Plan Verification by Exploiting Data Structures from HTN Planning. In *ECAI*.

Lin, S.; Behnke, G.; Ondrcková, 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*.

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

Lin, S.; Olz, C.; Helmert, M.; and Bercher, P. 2024. On the Computational Complexity of Plan Verification,(Bounded) Plan-Optimality Verification, and Bounded Plan Existence. In *AAAI*.

Masoumi, A.; Antoniazzi, M.; and Soutchanski, M. 2015. Modeling Organic Chemistry and Planning Organic Synthesis. In *GCAI*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - the planning domain definition language. Technical report.

Ondrcková, S.; Barták, R.; Bercher, P.; and Behnke, G. 2023. On the Impact of Grounding on HTN Plan Verification via Parsing. In *ICAART*.

Pantůčková, K.; and Barták, R. 2023. Using Earley Parser for Recognizing Totally Ordered Hierarchical Plans. In *ECAI*.

Rintanen, J. 2004. Complexity of Planning with Partial Observability. In *ICAPS*.

Savitch, W. J. 1970. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4.

Seipp, J.; and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In *ICAPS*.

Wichlacz, J.; Torralba, A.; and Hoffmann, J. 2019. Construction-planning models in minecraft. In *HPLAN*.