Was Fixing This *Really* That Hard? On the Complexity of Correcting HTN Domains

Songtuan Lin, Pascal Bercher

School of Computing, The Australian National University
{firstName.lastName}@anu.edu.au

Abstract

Automated modeling assistance is indispensable to the AI planning being deployed in practice, notably in industry and other non-academic contexts. Yet, little progress has been made that goes beyond smart interfaces like programming environments. They focus on autocompletion, but lack intelligent support for guiding the modeler. As a theoretical foundation of a first step towards this direction, we study the computational complexity of correcting a flawed Hierarchical Task Network (HTN) planning domain. Specifically, a modeler provides a (white) list of plans that are supposed to be solutions, and likewise a (black) list of plans that shall not be solutions. We investigate the complexity of finding a set of (optimal or suboptimal) model corrections so that those plans are (respective not) solutions to the corrected model. More specifically, we factor out each hardness source that contributes towards $\mathbb{NP}$-hardness, including one that we deem important for many other complexity investigations that go beyond our specific context of application. All complexities range between $\mathbb{NP}$ and Σ_2^p, raising the hope for efficient practical tools in the future.

1 Introduction

AI planning shows great potential for many application areas outside of academia like automated factories (Helmert and Lasinger 2010), robotics (Karpas and Magazzeni 2020), or assistance systems (Bercher et al. 2021; Grover et al. 2020). Despite the many advantages automated planning brings with it, we still don't see a large-scale application in industry. We believe that one of the biggest obstacles in AI planning becoming applied more broadly in practice is to model the respective application domain in the first place. In order to make planning more accessible, tools have been developed for modeling assistance, e.g., the successful online PDDL editor Planning.Domains. However, most of the tools like this only aim at providing a user-friendly programming environment, e.g., via syntax checking (Strobel and Kirsch 2020), visualization (Roberts et al. 2021), or interactive user interfaces (Vaquero et al. 2012; Viola et al. 2019).

To the best of our knowledge only a few approaches exist that provide further AI-based support. Lindsay et al. (2020), for example, refined an inaccurate hybrid domain to capture the environment more accurately, and Sreedharan et al. (2020) revised a dialogue domain via model reconciliation (Sreedharan, Chakraborti, and Kambhampati 2021; Sreedharan, Bercher, and Kambhampati 2022). Also in the context of modeling assistance Lin and Bercher (2021) studied the computational complexity of finding corrections to a flawed domain model provided a plan that is supposed to be a solution but currently (in the flawed model) is not. Later, Lin, Grastien, and Bercher (2023) developed a practical approach for solving this problem in classical planning. Such an example-guided methodology has been widely used in many disciplines, e.g., in theory revision (Greiner 1999), and in explaining or correcting the unsolvability of a planning problem (Göbelbecker et al. 2010; Gragera, García-Olaya, and Fernández 2022) – the latter two can be regarded contributions to AI planning modeling assistance if integrated into such a tool.

As another step towards such advanced modeling support, we investigate theoretical foundations for correcting flawed planning models in Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1996; Bercher, Alford, and Höller 2019). For this, we follow and extend the approach by Lin and Bercher (2021) where a plan serves as a test case to witness the correctness of a planning domain. More precisely, we follow the methodology of modifying a planning domain by *preserving* a set of *white* list plans while *rejecting* a set of *black* list plans. So, given a potentially flawed planning domain, a set of white list plans, and a set of black list plans, we want to output a sequence of change operations to the domain such that all plans in the white list are solutions in the *modified* domain and all plans in the black list are not.

Our contribution is twofold. First, we clarify every hardness source that makes correcting an HTN problem to preserve a set of *white list* plans $\mathbb{NP}$-hard in *partial order* HTN planning. This generalizes the results by Lin and Bercher (2021) which are restricted to *total order* (TO) HTN planning with only *one* white list plan being given. They also do not investigate the different causes for hardness, which we identify. Most importantly, one hardness source we found, namely that deciding whether a task sequence is a valid linearisation of a (partially ordered) task network in $\mathbb{NP}$-complete, is of great relevance to many other disciplines involving partial order, e.g., HTN plan verification or partial order causal link (POCL) planning, and hence can serve as a basis for complexity studies in those fields as well. Sec-

ond, we provide lower and upper complexity bounds for the problem when considering both a white list and a black list.

2 HTN Planning

We first introduce the HTN planning formalism used in the paper which is based upon a combination of the ones given by Bercher, Alford, and Höller (2019) and by Geier and Bercher (2011). We first give the definition of task networks.

A *task network* tn is a tuple $(T, \prec, \alpha)$ where T is a set of task identifiers, $\prec \subseteq T \times T$ specifies the partial order defined over T, and α is a function that maps each task identifier to the respective task name.

A *linearisation* of a task network $tn = (T, \prec, \alpha)$, $\overline{tn}$, is a total order of T which respects $\prec$. We use the notation $\alpha(\overline{tn})$ to refer to the task sequence represented by the linearisation, i.e., if $\overline{tn} = \langle t_1 \cdots t_n \rangle$, then $\alpha(\overline{tn}) = \langle \alpha(t_1) \cdots \alpha(t_n) \rangle$.

The task names in a task network are further categorized as being primitive or compound. A *primitive task name* p, also called an *action*, is mapped to its precondition, add, and delete list, each of which is a set of propositions, by a function δ written $\delta(p) = (prec, add, del)$, where add together with del is called the effects of p. We also write $prec(p)$, $add(p)$, and $del(p)$ for short. On the other hand, a compound task name c can be refined (decomposed) into a task network tn by some *method* $m = (c, tn)$.

An action sequence $\langle p_1 \cdots p_n \rangle$ is executable in a *state*, which is also a set of propositions, if there exists a sequence of states $\langle s_0 \cdots s_n \rangle$ such that $s_0 = s$, and for each $1 \leq i \leq n$, $s_{i-1} \subseteq prec(p_i)$ and $s_i = (s_{i-1} \backslash del(p_i)) \cup add(p_i)$.

Two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ are said to be *isomorphic*, written $tn \overset{\varphi}{\cong} tn'$, *iff* there exists a one-to-one mapping $\varphi : T \to T'$ such that for all $t \in T$, $\alpha(t) = \alpha'(\varphi(t))$, and for all $t_1, t_2 \in T$, if $(t_1, t_2) \in \prec$, then $(\varphi(t_1), \varphi(t_2)) \in \prec'$. We also write $tn \cong tn'$ if the mapping φ is not explicitly required in contexts.

Given a task network tn, the notations $T(tn)$, $\prec(tn)$, and $\alpha(tn)$ refer to the task identifier set, the partial order, and the identifier-name mapping function of tn, respectively.

For convenience, we define a *restriction operation*. Let D and V be two arbitrary sets, $R \subseteq D \times D$ a relation, $f : D \to V$ a function, and tn a task network. The restrictions of R and f to some set X are defined by

- $R|_X = R \cap (X \times X)$
- $f|_X = f \cap (X \times V)$
- $tn|_X = (T(tn) \cap X, \prec(tn)|_X, \alpha(tn)|_X)$

An *HTN planning problem* P is a tuple (D, tn_I, s_I) where D is the domain of P. D is a tuple (F, N_p, N_c, δ, M) where F is a finite set of propositions, N_p is a finite set of primitive task names, N_c is a finite set of compound task names with $N_c \cap N_p = \emptyset$, $\delta : N_p \to 2^F \times 2^F \times 2^F$ maps primitive task names to their preconditions and effects, and M is a set of (decomposition) methods. tn_I is the initial task network, and $s_I \in 2^F$ is the initial state.

Let $tn = (T, \prec, \alpha)$ be a task network, $t \in T$ be a task identifier, c be a compound task name with $(t, c) \in \alpha$, and $m = (c, tn_m)$ be a method. We say m *decomposes* tn into another task network $tn' = (T', \prec', \alpha')$, written $tn \to_m tn'$, *iff* there exists a task network $tn'_m = (T_m, \prec_m, \alpha_m)$ with $tn'_m \cong tn_m$ such that

- $T' = (T \backslash \{t\}) \cup T_m$.
- $\prec' = (\prec \cup \prec_m \cup \prec_X)|_{T'}$ with $\prec_X = \{(t_1, t_2) \,|\, (t_1, t) \in \prec, t_2 \in T_m\} \cup \{(t_2, t_1) \,|\, (t, t_1) \in \prec, t_2 \in T_m\}$.
- $\alpha' = (\alpha \backslash \{(t, c)\}) \cup \alpha_m$.

A task network tn is decomposed into another task network tn' by a sequence of methods $\overline{m} = \langle m_1 \cdots m_n \rangle$ with $n \in \mathbb{N}_0$ ($\mathbb{N}_0 = \mathbb{N} \cup \{0\}$), written $tn \to^*_{\overline{m}} tn'$, *iff* there exists a sequence of task networks $\langle tn_0 \cdots tn_n \rangle$ such that $tn_0 = tn$, $tn_n = tn'$, and for each $1 \leq i \leq n$, $tn_{i-1} \to_{m_i} tn_i$.

A solution to an HTN planning problem $P = (D, tn_I, s_I)$ is a task network tn such that all tasks in it are primitive, there is a method sequence $\overline{m}$ with $tn_I \to^*_{\overline{m}} tn$, and it possesses a linearisation $\overline{tn}$ with $\alpha(\overline{tn})$ being executable in s_I.

Notably, in this paper, we consider correcting HTN domain models by providing white list and black list *plans*. A plan refers to a sequence of actions which is thus different from a solution to an HTN planning problem syntactically. In practice, a human is usually more interested in an executable plan rather than a partial order task network which is the reason why we consider plans instead of solution task networks (which have a plan as witness).

We thus extend the term 'solution' and define the criteria for a plan being a solution to a planning problem as follows.

Definition 1. Let P be an HTN planning problem. A plan π is a solution to P (thus overriding our previous definition) *iff* π is executable in s_I, and there is a task network tn which is a solution to P and has a linearisation $\overline{tn}$ with $\pi = \alpha(\overline{tn})$.

3 Model Change Operations

We now define four change operations for changing methods in an HTN planning domain, i.e., adding/deleting actions and adding/deleting ordering constraints between actions. We start with the one which adds an action to a method.

Definition 2. Let $m = (c, tn)$ with $tn = (T, \prec, \alpha)$ be a method, $T_a = \{t_1, \cdots, t_n\}$ and $T_b = \{t'_1, \cdots t'_m\}$ with $n, m \in \mathbb{N}_0$ and $T_a \cap T_b = \emptyset$ be two subsets of T, and $p \in N_p$ be a primitive task name. The operation ACT$^+$ is a function that takes as inputs m, T_a, T_b, and p and outputs a new method $m' = (c, tn')$ with $tn' = (T', \prec', \alpha')$ such that $T' = T \cup \{t\}$ with $t \notin T$ being a new task identifier, $\prec' = (\prec \cup \prec_a \cup \prec_b)^{+1}$ with $\prec_a = \bigcup_{i=1}^{n} \{(t_i, t)\}$ and $\prec_b = \bigcup_{i=1}^{m} \{(t, t'_i)\}$, and $\alpha' = \alpha \cup \{(t, p)\}$.

Informally, the operation inserts an action to a position in tn that is after the tasks listed in T_a and before those in T_b. For instance, a new action is placed before all tasks in tn if $T_a = \emptyset$ and $T_b = T$.

When removing an action from some method, all ordering constraints associated with this action should be removed.

Definition 3. Let $m = (c, tn)$ with $tn = (T, \prec, \alpha)$ be a method, and $t \in T$ be a task identifier. The operation ACT$^-$ is a function that takes as inputs m and t and outputs a new method $m' = (c, tn')$ with $tn' = tn|_{T \backslash \{t\}}$.

[1] The superscript $^+$ refers to the transitive closure.

The operations aiming at changing ordering constraints between actions in a method are defined as follows.

Definition 4. Let $m = (c, tn)$ with $tn = (T, \prec, \alpha)$ be a method and $t_1, t_2 \in T$ two identifiers with $\alpha(t_1), \alpha(t_2) \in N_p$. The operation ORD$^+$ is a function that takes as inputs m and (t_1, t_2) and outputs a new method $m' = (c, tn')$ with $tn' = (T', \prec', \alpha')$ such that $T' = T$, $\prec' = (\prec \cup \{(t_1, t_2)\})^+$, and $\alpha' = \alpha$.

Definition 5. Let $m = (c, tn)$ with $tn = (T, \prec, \alpha)$ be a method and $t_1, t_2 \in T$ two identifiers with $\alpha(t_1), \alpha(t_2) \in N_p$ and $(t_1, t_2) \in \prec$. The operation ORD$^-$ is a function that takes as inputs m and (t_1, t_2) and outputs a new method $m' = (c, tn')$ with $tn' = (T', \prec', \alpha')$ such that $T' = T$, $\prec' = \prec \backslash \{(t_1, t_2)\}$, and $\alpha' = \alpha$.

Given two methods m and m', we write $m \to_\triangle m'$ where $\triangle$ refers to one of the operations defined previously with certain parameters if m' can be obtained from m via performing $\triangle$. We further write $P \to_\triangle P'$ if P and P' are two HTN planning problems which respectively contain m and m', and P and P' differ solely in these two methods.

Lastly, we present the definition of model transformations on an HTN domain, generalizing the one in the TO setting given by Lin and Bercher (2021) to the PO setting.

Definition 6. Given an HTN planning problem P and a change sequence $\mathbf{\Delta} = \langle \triangle_1 \cdots \triangle_n \rangle$ ($n \in \mathbb{N}_0$), we say P is transformed into another planning problem P' by $\mathbf{\Delta}$, written $P \to^*_{\mathbf{\Delta}} P'$ if there exists a sequence of planning problems $\langle P_0 \cdots P_n \rangle$ such that $P_0 = P$, $P_n = P'$, and $P_{i-1} \to_{\triangle_i} P_i$ for each $1 \leq i \leq n$. Particularly, we define $P \to^*_{\mathbf{\Delta}} P$ if $\mathbf{\Delta}$ contains no operations. We will use $|\mathbf{\Delta}|$ to denote the length of a change sequence $\mathbf{\Delta}$ throughout the paper.

For any two methods m and m', we write $m \to^*_{\mathbf{\Delta}} m'$ if there is a sequence of methods $\langle m_0 \cdots m_n \rangle$ such that $m = m_0$, $m' = m_n$, and for each $1 \leq i \leq n$, $m_{i-1} \to_{\triangle_i} m_i$.

One remark is that if P can be transformed into P', there must exist a one-to-one correspondence between the methods in these two planning problems.

Proposition 1. *Given two planning problems P and P' with M and M' being their method sets respectively, if $P \to^*_{\mathbf{\Delta}} P'$ for some change sequence $\mathbf{\Delta} = \langle \triangle_1 \cdots \triangle_n \rangle$ ($n \in \mathbb{N}_0$), there must exist a bijective mapping $\beta_{\mathbf{\Delta}} : M \to M'$ such that for each $m \in M$, $m \to^*_{\mathbf{\Delta}_j} \beta_{\mathbf{\Delta}}(m)$ for some subsequence $\mathbf{\Delta}_j = \langle \triangle_{i_1} \cdots \triangle_{i_j} \rangle$ with $1 \leq i_1 \leq \cdots \leq i_j \leq n$.*

Notably, one might find that we did not define operations that change actions' preconditions and effects for dealing with the case where plans that are supposed to be solutions are *not* executable. The reason for leaving out this case is that the respective problem, i.e., making an unexecutable plan executable by changing actions, has already been studied by Lin and Bercher (2021), and changing actions is orthogonal to changing methods.

4 Complexity of Changing the Model – Given Only White List Plans

We now move on to investigate the computational complexity of deciding whether a set of white list plans can be turned into solutions via changing the domain model. This was first studied for TOHTN planning by Lin and Bercher (2021) with only one white list plan being concerned. We extend it to the partial order setting, and with a *set* of white list plans. We will also identify all three nondeterminism sources that result in $\mathbb{NP}$-hardness, see Fig. 1 for these sources.

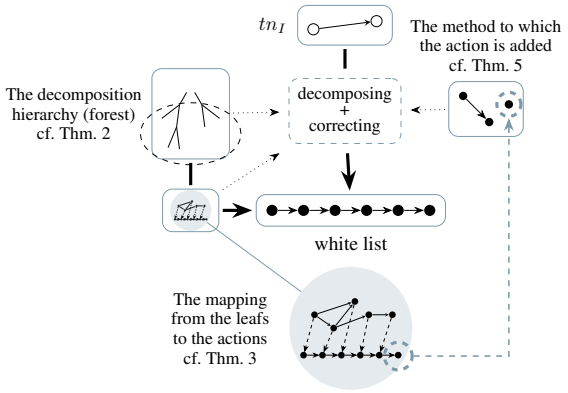

Figure 1: The three hardness sources making the problem of turning a white list plan into a solution by correcting the domain $\mathbb{NP}$-hard, namely, missing the decomposition hierarchy (forest), missing the mapping from the leafs of the forest to the actions in the plan, and missing the knowledge of the method to which the missing actions should be added.

Definition 7. Let $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^-\}$ and $|X| \geq 1$, P be a planning problem, and Π be a set of plans. We define the problem CHANGE$_X$ as: Is there a sequence of model change operations $\mathbf{\Delta}$ such that $P \to^*_{\mathbf{\Delta}} P'$, π is a solution to P' for each $\pi \in \Pi$, and $\mathbf{\Delta}$ consists of ACT$^+$,ACT$^-$, and ORD$^-$, according to X?

We use the subscript X to restrict change operations that are allowed in order to study whether performing some operations is computationally harder than others. Further, we do not consider ORD$^+$ in the problem because introducing more ordering constraints to a domain only increases the chance of a plan *not* being a solution. However, this operation will be required when accounting for black list plans.

We first show that the problem is in $\mathbb{NP}$ regardless what change operations are allowed. The basis for membership is the fact that if there exists a sequence of change operations that turns a set of plans into solutions, then there *must* exist one of length smaller than a polynomial. This fact is first proved by Lin and Bercher (2021) in TOHTN planning with *only one* white list plan being given. We now extend it to a *set* of white list plans with the partial order setting.

Lemma 1. *Let P and Π be the planning problem and the set of white list plans given by an instance of the CHANGE$_X$ problem with $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^-\}$ and $|X| \geq 1$. There must exist a change sequence $\mathbf{\Delta}$ consisting of operations restricted by X such that $P \to^*_{\mathbf{\Delta}} P'$, π is a solution to P' for all $\pi \in \Pi$, and $|\mathbf{\Delta}|$ is bounded by a polynomial, provided that any change sequence exists that meets the restriction of X and turns the plans in Π into solutions.*

Proof. We start by considering the case where all operations are allowed. The key observation here is that, for the *shortest* change sequence that turns all white list plans into solutions, the maximal number of action insertions is bounded by the maximal length of plans in the white list, otherwise, by the piegeonhole theorem, there must be an action that is added first and deleted afterwards, which contradicts the sequence being the shortest one. Similarly, the number of action deletions and of ordering constraint and deletions is respectively bounded by the total number of actions and of ordering constraints from all methods. Thus, the length of the sequence is bounded by the sum of those three numbers.

For other cases where *not* all change operations are allowed, the length of the shortest change sequence is strictly smaller. Thus, the lemma holds in general. □

We can then exploit this result to prove $\mathbb{NP}$ membership.

Theorem 1. *Let* $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^-\}$ *and* $|X| \geq 1$. CHANGE_X *is in* $\mathbb{NP}$.

Proof. For *every* $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^-\}$ and $|X| \geq 1$, we nondeterministically guess a change sequence of length smaller than or equal to the polynomial upper bound given by Lem. 1 which turns P into P'. We then employ the nondeterministic polynomial time algorithm VERIFYTN by Behnke, Höller, and Biundo (2015) to check whether for each $\pi \in \Pi$, π is a solution to P'. The entire procedure has polynomial complexity, CHANGE_X is thus in $\mathbb{NP}$. □

The first source of nondeterminism We now show that the problem is also $\mathbb{NP}$-hard disregarding what change operations are allowed. Notably, hardness of the variants with $\text{ACT}^+ \in X$ or $\text{ACT}^- \in X$ can be inferred directly from the proof given by Lin and Bercher (2021). However, their proof did not work for the case where only ORD^- is allowed. Thus, we first prove that the variant where only deleting ordering constraints is allowed is still $\mathbb{NP}$-hard. We believe that this is strongly counterintuitive because, at first glance, one might think that it can be easily decided by deleting all ordering constraints in all methods. Our proof shows that this is not the case because we face the hardness source of finding decomposition hierarchies that can lead to the white list plans.

Our proof relies on the reduction from the VERIFYSEQ problem (Behnke, Höller, and Biundo 2015). We reproduce its definition as follows.

Definition 8 (Behnke, Höller, and Biundo (2015, Def. 12))**.** Let P be an HTN planning problem and π an action sequence, the problem VERIFYSEQ is to decide whether π is a solution to P.

Theorem 2. CHANGE_X *with* $X = \{\text{ORD}^-\}$ *is* $\mathbb{NP}$*-hard.*

Proof. In particular, Behnke, Höller, and Biundo (2015, Cor. 5) shown that the VERIFYSEQ problem is $\mathbb{NP}$-hard even for *totally unordered* HTN planning problems where the initial task network together with every method has *no* ordering constraints. We can thus reduce a VERIFYSEQ instance with an unordered HTN planning problem P and a plan π as inputs to a CHANGE_X instance with $X = \{\text{ORD}^-\}$ by keeping P unchanged and letting the white list plan set contains solely the plan π. Since P is already unordered, the operation ORD^- is redundant. Hardness follows immediately. □

We then have the following corollary by consulting previous results by Lin and Bercher (2021).

Corollary 1. *Let* $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^-\}$ *and* $|X| \geq 1$. CHANGE_X *is* $\mathbb{NP}$*-hard.*

Further, if we take a closer look at the proof by Behnke, Höller, and Biundo (2015) for hardness of the VERIFYSEQ problem, we could observe that the authors constructed a reduction from the vertex cover problem where a vertex cover problem instance has a cover of size smaller than or equal to some integer k *iff* the constructed HTN planning problem has a decomposition hierarchy that leads to the constructed plan. This implies that finding decomposition hierarchies is thus a hardness source for CHANGE_X.

The second source of nondeterminism Next we investigate whether the problem becomes easy if decomposition hierarchies are given which are supposed to result in the plans in the white list. This is unfortunately not the case because of the second hardness source – finding the mapping between a plan and the result of a decomposition hierarchy.

Concretely, we consider the scenario here in which a domain modeler provides not only white list plans but decomposition hierarchies for each plan in the white list which is supposed to lead to it (the plan). Note that, in order to make this scenario practical, we assume that these decomposition hierarchies are constructed in the potentially *flawed* domain. Lin and Bercher (2021) formulated a special case of this scenario in TOHTN planning as a decision problem where only *one* white list plan is given by using a method sequence to capture a decomposition hierarchy, and the authors proved that the decision problem is $\mathbb{NP}$-hard.

However, using a method sequence to represent a decomposition hierarchy introduces an unexpected nondeterminism source if some compound task occurs more than once in the decomposition process for the reason that such a compound task could be refined by any method in the sequence which can decompose it. For instance, consider a task network $(\{t_1, t_2\}, \{(t_1, t_2)\}, \alpha)$ such that $\alpha(t_1) = \alpha(t_2) = c$ with c being some compound task and a method sequence $(c, tn_1)\,(c, tn_2)$ in which $tn_1 \not\cong tn_2$ are two task networks. Clearly, this method sequence can decompose the task network into two different ones depending on how each method is matched to a compound task.

This additional nondeterminism also makes the problem studied by Lin and Bercher (2021) a little unrealistic because when a domain modeler specifies a decomposition hierarchy, the matching between each method and each compound task is likely known in advance.

We thus adapt a rigorous representation of decomposition hierarchies, called *decomposition trees*, initially developed by Geier and Bercher (2011), which specify the matching between a compound task and a method which decomposes it. Here, we employ the formalism by Alford et al. (2014) and by Höller et al. (2020) and extend it by letting a decomposition process start with an initial task network instead of an initial task, and it is thus called a *decomposition forest*.

Given a planning problem P, a decomposition forest $g = (V, E, \prec_g, \alpha_g, \beta_g)$ with respect to P is a set of labeled directed trees where V and E are the sets of vertices and edges respectively, $\prec_g$ is a partial order defined over V, $\alpha_g : V \to N_p \cup N_c$ labels a vertex with a task name, and β_g maps a vertex $v \in V$ to a method $(c, tn) \in M$ and an isomorphism relation φ from $T(tn)$ to the children of v written $ch(v)$, i.e., $\varphi : T(tn) \to ch(v)$.

A decomposition forest is *valid iff* for each $t \in T(tn_I)$, there exists a root vertex $r \in V$ labeled with $\alpha(tn_I)(t)$, and for each $v \in V$ with $\beta_g(v) = (m, \varphi)$, $m = (c, tn)$, and $c \in N_c$, the following holds.

1) $\alpha_g(v) = c$.
2) tn is isomorphic to the task network induced by $ch(v)$ under the isomorphism relation φ, i.e.,
$$tn \overset{\varphi}{\cong} (ch(v), \prec_g|_{ch(v)}, \alpha_g|_{ch(v)})$$
3) For any child v_c of v and any $v' \in V$, if $(v', v) \in \prec_g$, $(v', v_c) \in \prec_g$, and if $(v, v') \in \prec_g$, $(v_c, v') \in \prec_g$.
4) No other ordering constraints exist in $\prec_g$ except those demanded by 2) and 3).

The yield of a decomposition forest g, $yield(g)$, is the task network consisting of the leafs of g together with the associated ordering constraints and the labels, i.e., $yield(g) = (L(g), \prec_g|_{L(g)}, \alpha_g|_{L(g)})$ where $L(g)$ refers to the set of all leafs of g. One can easily observe that a plan is a solution to an HTN planning problem *iff* it can be viewed as a valid linearisation of the yield of a valid decomposition forest.

We now show that correcting an HTN domain is $\mathbb{NP}$-hard even if a set of decomposition forests are given which are supposed to lead to the white list plans. This result is a stronger version of the one by Lin and Bercher (2021) because 1) we take as an input a decomposition forest instead of a method sequence, and 2) it holds even if only *one* decomposition forest and *one* plan are given. We also explore the hardness source causing this along the way which could serve as a foundation for complexity investigations in many disciplines beyond HTN planning. We first formulate the respective decision problem in terms of decomposition forests.

Definition 9. Given an HTN planning problem P, a set of plans $\Pi = \{\pi_1, \cdots, \pi_n\}$, and a set of decomposition forests $\mathcal{G} = \{g_1, \cdots, g_n\}$ with respect to P where $g_i = (V_i, E_i, \prec_{g_i}, \alpha_{g_i}, \beta_{g_i})$ for each $1 \leq i \leq n$, we define the problem CHANGEDF$_X$ with $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^-\}$ and $|X| \geq 1$ as: Is there a change sequence $\mathbf{\Delta}$ consisting of operations restricted by X such that $P \to^*_{\mathbf{\Delta}} P'$, and for each $1 \leq i \leq n$, there exists a valid decomposition forest $g'_i = (V'_i, E'_i, \prec_{g'_i}, \alpha_{g'_i}, \beta_{g'_i})$ with respect to P' such that the following holds:

1) $yield(g'_i)$ has a linearisation $\overline{tn}$ with $\pi_i = \alpha(\overline{tn})$,
2) $|\beta_{g'_i}| = |\beta_{g_i}|$, and
3) for all $v \in V_i$, if $\alpha_{g_i}(v) \in N_c$ and $\beta_{g_i}(v) = (m, \varphi)$ for some $m \in M$ and isomorphic relation φ, then $v \in V'_i$ and $\beta_{g'_i}(v) = (\beta_{\mathbf{\Delta}}(m), \varphi')$ such that for all $v^* \in ch(v) \cap ch(v')$, $\varphi^{-1}(v^*) = {\varphi'}^{-1}(v^*)$[2].

[2] φ^{-1} and ${\varphi'}^{-1}$ represent the inverse mapping of φ and φ'.

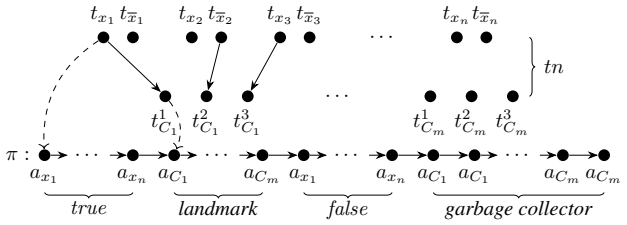

Figure 2: The constructions of tn and π in the proof of Thm. 3. The solid arrows represent ordering constraints. The dashed arrows illustrate the mapping.

Notably, each decomposition forest given in the set $\mathcal{G}$ is constructed with respect to the potentially flawed model P. The restriction on g'_i for each $1 \leq i \leq n$ basically demands that each compound task in g'_i being decomposed as well as the method that decomposes it are identical to those in g_i, which thus encodes the criterion that π_i should be a solution under the respective decomposition hierarchy.

Hardness of this problem adheres to the fact that deciding whether an action sequence is a valid linearisation of a task network is $\mathbb{NP}$-complete (the second hardness source). This problem, called VALIDSEQ, is formulated as follows.

Definition 10. Let tn be a *primitive* task network and π be an action sequence. The problem VALIDSEQ is to decide whether there exists a linearisation $\overline{tn}$ of tn with $\pi = \alpha(\overline{tn})$.

Theorem 3. VALIDSEQ *is NP-complete.*

Proof. Membership: Given a (partial order) task network tn and an action sequence π, membership can be easily proved by first guessing a linearisation $\overline{tn}$ of tn and then verifying whether $\alpha(\overline{tn}) = \pi$.

Hardness: We reduce from the NP-complete 3SAT problem (Cook 1971). Let $C_1, \cdots, C_m$ be the clauses of a SAT formula given by an 3SAT instance which consists of variables $x_1, \cdots x_n$. We construct one action a_{x_i} for each variable x_i ($1 \leq i \leq n$) and one action a_{C_j} for each clause C_j ($1 \leq j \leq m$). The key idea of the reduction is constructing a task network $tn = (T, \prec, \alpha)$ encoding the constraints imposed by the SAT formula.

To this end, we construct $t_{x_i}, t_{\overline{x}_i} \in T$ with $\alpha(t_{x_i}) = \alpha(t_{\overline{x}_i}) = a_{x_i}$ for each $1 \leq i \leq n$ and $t^1_{C_j}, t^2_{C_j}, t^3_{C_j} \in T$ with $\alpha(t^1_{C_j}) = \alpha(t^2_{C_j}) = \alpha(t^3_{C_j}) = a_{C_j}$ for each $1 \leq j \leq m$. An ordering constraint $(t_{x_i}, t^k_{C_j})$ or $(t_{\overline{x}_i}, t^k_{C_j})$ exists for some $1 \leq k \leq 3$ if the literal x_i or $\overline{x}_i$ is in the clause C_j. Fig. 2 depicts an example of such construction in which C_1 consists of the literals x_1, $\overline{x}_2$, and x_3. Lastly, we construct the action sequence π as shown by the Fig. 2.

A literal x_i or $\overline{x}_i$, $1 \leq i \leq n$, is set to true if the respective task identifier t_{x_i} or $t_{\overline{x}_i}$ is mapped to the action a_{x_i} placed in the subsequence labeled with *true* (and thus it is false if the task identifier is mapped to a_{x_i} in the subsequence labeled with *false*). The presence of the subsequence labeled by *landmark* enforces that an action a_{C_j} ($1 \leq j \leq m$) in it must map to a task identifier $t^k_{C_j}$ for some $1 \leq k \leq 3$. For such a mapping to be valid, $t^k_{C_j}$'s preceding task must

be mapped to an action in $true$, which thus encodes the criterion that for each clause, at least one literal in it should be true. For instance, in Fig. 2, $t^1_{C_1}$ can be mapped to a_{C_1} in $landmark$ if t_{x_1} is mapped to a_{x_1} in $true$. The remaining tasks can then be mapped to the actions in the subsequence labeled by *garbage collector* without violating any ordering constraint. Lastly, we prove that the given 3SAT instance has a *yes* answer if and only if the VALIDSEQ instance constructed has one.

($\Longrightarrow$): Suppose there exists a truth assignment over the variables $x_1, \cdots, x_n$ which satisfies the given SAT formula. The constructed action sequence can form a valid linearisation of the task network in the sense that every pair of the task identifiers, t_{x_i} and $t_{\overline{x}_i}$, are mapped to the respective actions in *true* and *false* according to the truth assignment, and each action a_{C_i} ($1 \leq i \leq m$) in *landmark* is mapped to a task identifier $t^j_{C_i}$ ($1 \leq j \leq 3$) whose preceding task is mapped to an action in *true*. For instance, in Fig. 2, the action a_{C_1} in *landmark* is mapped to the task $t^1_{C_1}$ because t_{x_1} is mapped to a_{x_1} in *true*. The remaining tasks in the task network are then accordingly mapped to the actions in *garbage collector*, and no ordering constraints are violated.

($\Longleftarrow$): Suppose there exist no truth assignments satisfying the SAT formula. For any truth assignment, there must be some clause C_j ($1 \leq j \leq m$) where all three literals are set to false, which means that the tasks representing these three literals will be mapped to the respective actions in *false*. Thus, at least one ordering constraint is violated. □

Clearly, for *any* $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^-\}$ with $|X| \geq 1$, we can easily reduce the VALIDSEQ problem to the respective CHANGEDF$_X$ problem by letting the primitive task network given by the VALIDSEQ problem be the initial task network, and the action sequence be the *only* white list plan.

Corollary 2. *Let* $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^-\}$ *and* $|X| \geq 1$, CHANGEDF$_X$ *is* $\mathbb{NP}$*-hard.*

One could observe that the VALIDSEQ problem is equivalent to asking whether there is a bijective mapping between the actions in the task network and in the plan. It follows that the second hardness source of correcting an HTN domain is finding the mapping between the yield of a decomposition forest and a plan.

We now discuss why we believe that the $\mathbb{NP}$-completeness result for deciding VALIDSEQ (Thm. 3) is important for future complexity investigations as well, i.e., in a general context of HTN planning and where reasoning about partially ordered plans is of importance, such as in partial order causal link (POCL) planning.

For the latter, note that it was shown that deciding whether an executable linearization of a partially ordered plan *exists* is $\mathbb{NP}$-complete (Nebel and Bäckström 1994; Erol, Hendler, and Nau 1996). Now we also know that even providing such a (non-labeled) sequence does not make that task easier.

The VALIDSEQ problem can be observed as a fundamental source of hardness of many (related) problems in HTN planning (and specifically those of verification). E.g., NP-hardness of the VERIFYSEQ problem (cf. Def. 8) (Behnke, Höller, and Biundo 2015) follows directly from our result. This is also the case for $\mathbb{NP}$-hardness of the PLANCOMPATIBILITY problem (Behnke, Höller, and Biundo 2015), which asks whether a (partially ordered) task network is an ordering refinement of another given task network. Both proofs required one page each, but easily follow from ours.

We can also conclude from this result that HTN plan verification remains $\mathbb{NP}$-hard even if in addition to the task network to verify we have an executable action sequence provided. This thus generalizes the respective result (Cor. 1) by Behnke, Höller, and Biundo (2015). So we expect that our result will be convenient in many complexity investigations where the verification of task networks plays some role.

The third source of nondeterminism Lastly, we study the problem of correcting HTN domains under the strongest assumption where the correspondence between the tasks in the yield of a decomposition forest and the actions in a plan is also provided.

More concretely, this refers to the scenario where after a domain modeler provides a white list plan set and the respective decomposition forest set, the correspondence between the tasks in a decomposition forest and the actions in a plan is observed by the modeler as well. The modeler could thus identify which action in the plan is missing from the decomposition forest and which is not occurred in the plan but is appeared in the hierarchy. Our goal is again to turn the white list plans into solutions by considering all the information.

To formulate this scenario, we capture the correspondence between a decomposition forest g and a plan by a function $\varrho : T(yield(g)) \to \mathbb{N}$ which maps a task in the yield of g to an *index* of the plan. We call such a ϱ a *modification function* of which the syntax and semantics are defined as follows:

Definition 11. Let $\pi = \langle a_1 \cdots a_n \rangle$ a plan and g a decomposition forest. $\varrho : T(yield(g)) \to \mathbb{N}$ is a modification function if ϱ is *one-to-one* (i.e., for any $t_1, t_2 \in T(yield(g))$, $\varrho(t_1) = \varrho(t_2)$ *iff* $t_1 = t_2$), and for each $t \in T(yield(g))$, one of the following holds: either $\varrho(t) = i$ for some $1 \leq i \leq n$ with $\alpha(yield(g))(t) = a_i$, or $\varrho(t)$ is undefined.

For any $t \in T(yield(g))$, the semantics of ϱ is defined as:

- If $\varrho(t) = i$ for some $1 \leq i \leq n$, t is supposed to be the action a_i in π, i.e., t is inherited to a_i.
- If $\varrho(t)$ is undefined, t is supposed to be deleted from g.
- For some $1 \leq i \leq n$, if there is *not* a $t \in T(yield(g))$ with $\varrho(t) = i$, a_i in π is supposed to be added to g.

Let P be a planning problem, $g = (V, E, \prec_g, \alpha_g, \beta_g)$ a decomposition forest, $\pi = \langle a_1 \cdots a_n \rangle$ a plan, and ϱ a modification function with respect to g and π. Suppose $\mathbf{\Delta}$ is a change sequence such that $P \to^*_{\mathbf{\Delta}} P'$, and there is a valid decomposition forest $g' = (V', E', \prec_{g'}, \alpha_{g'}, \beta_{g'})$ satisfying *all* three criteria given in Def. 9. Let $\overline{tn} = \langle t_1 \cdots t_n \rangle$ be the linearisation of $yield(g')$ with $\alpha(\overline{tn}) = \pi$. We say $\mathbf{\Delta}$ *preserves* the function ϱ if the following holds: For every *leaf* $l \in V$ and its parent $v \in V$ with $\beta_g(v) = (m, \varphi)$, if $\varrho(l) = i$ for some a_i in π, and there exists a leaf $l' \in V'$ such that $v \in V'$ is also the parent of l', $\beta_{g'}(l') = (\beta_{\mathbf{\Delta}}(m), \varphi')$, and $\varphi^{-1}(l) = {\varphi'}^{-1}(l')$, then $t_i = l'$.

We now formulate the decision problem we will study:

Definition 12. Given an HTN planning problem P, a set

of plans $\Pi = \{\pi_1, \cdots, \pi_n\}$, a set of decomposition forests $\mathcal{G} = \{g_1, \cdots, g_n\}$ with respect to P, and a set of modification functions $\Phi = \{\varrho_1, \cdots, \varrho_n\}$ with respect to π_i and g_i for each $1 \leq i \leq n$, we define the problem CHANGEDFM$_X$ in the same way as CHANGEDF$_X$ except that we demand that the change sequence $\mathbf{\Delta}$ should preserve ϱ_i for each i.

Clearly, CHANGEDFM$_X$ with ACT$^+ \notin X$ can be solved in polynomial time due to the presence of modification functions. More concretely, for each g_i and ϱ_i, we could find all $t \in yield(g_i)$ with $\varrho_i(t)$ being undefined and deciding whether t could be removed from the respective method by tracing backward through g_i. In particular, the task t can be deleted from a method m if for all other occurrence of m in *all* g_j with $1 \leq j \leq n$, deleting t will *not* violate the function ϱ_j. The similar approach can then be used to decide whether a plan π_i can be turned into a valid linearisation by deleting ordering constraints from methods.

Theorem 4. CHANGEDFM$_X$ *is in* $\mathbb{P}$ *if* ACT$^+ \notin X$.

Unfortunately, the problem is still $\mathbb{NP}$-hard if ACT$^+$ is allowed. This stems from the third hardness source which is to find which method should a missing action being added to. This hardness source follows from the proof by Lin and Bercher (2021) for showing CHANGE$_X$ with $X = \{$ACT$^+\}$ is $\mathbb{NP}$-hard in TOHTN in which the authors constructed a reduction from the independent set problem such that an independent set of size k ($k \in \mathbb{N}$) can be found in a graph *iff* a plan can be turned into a solution by finding some methods to which missing actions are added. Further, in their construction, all other operations including ORD$^-$ are redundant.

Theorem 5. CHANGEDFM$_X$ *is* $\mathbb{NP}$*-hard if* ACT$^+ \in X$.

Taking into account Thm. 1, we have the following.

Corollary 3. *Let* $X \subseteq \{$ACT$^+$, ACT$^-$, ORD$^-\}$ *and* $|X| \geq 1$, CHANGE$_X$ *and* CHANGEDF$_X$ *are* $\mathbb{NP}$*-complete. For any* X *with* ACT$^+ \in X$, CHANGEDFM$_X$ *is* $\mathbb{NP}$*-complete, otherwise, it is in* $\mathbb{P}$.

In conclusion, we have now identified all three hardness sources result in $\mathbb{NP}$-hardness of correcting an HTN domain, namely, finding decomposition forests, finding mappings between yields of decomposition forests and plans, and finding methods to which missing actions should be added.

Optimal Change Sequences It is quite often the case in practice that a domain modeler might be interested in finding a minimal length change sequence instead of an arbitrary one which turns a set of plans into solutions. We formulate such a scenario by introducing an additional integer k.

Definition 13. Let $k \in \mathbb{N}$ and $X \subseteq \{$ACT$^+$, ACT$^-$, ORD$^-\}$ and $|X| \geq 1$. We define CHANGE^k_X, CHANGEDF^k_X, and CHANGEDFM^k_X as the problems in which the demanded change sequence is of length smaller or equal to k.

It follows from Lem. 1 that the shortest change sequence is bounded by a polynomial in length. Thus, the presented k-bounded problems are in $\mathbb{NP}$ because no matter how large the given k is, we can always guess and verify a change sequence of polynomial length. Further, we can easily reduce an unbounded problem into the respective k-bounded one by letting k be the polynomial upper bound.

Corollary 4. *Given* $k \in \mathbb{N}$, CHANGE^k_X *and* CHANGEDF^k_X *are* $\mathbb{NP}$*-complete for any* $X \subseteq \{$ACT$^+$, ACT$^-$, ORD$^-\}$ *and* $|X| \geq 1$. CHANGEDFM^k_X *is* $\mathbb{NP}$*-complete if* ACT$^+ \in X$, *otherwise, it is in* $\mathbb{P}$.

Exploiting the hardness sources We now want to discuss how practical realizations of solving the CHANGE$_X$ problem can be developed in light of our results.

Firstly, since the problem CHANGE$_X$ is $\mathbb{NP}$-complete, it is thus natural to think of encoding it as a SAT formula and employing a SAT solver to solve the problem. For this, we have to encode all nondeterminism sources. Having each source identified in advance could thus help us understand what we are facing and adapt some existing SAT encoding for some problem that is related or similar to a hardness source. More concretely, Behnke, Höller, and Biundo (2017; 2018; 2019) proposed several approaches for encoding a decomposition forest as a SAT formula, which could thus be adapted here for coping with the first hardness source.

Those SAT encoding of decomposition forests can be easily extended to incorporate action deletions and insertions, i.e., the third hardness source. More concretely, for an action a and a method m, we can define a variable I^a_m to indicate whether a is inserted to m. Similarly, for an action a in a method m, we could define another variable R^a_m indicating whether a is removed from m. The consequence of adding or deleting an action can then be incorporated into the SAT formula for the decomposition forest by constructing further implications defined over those variables.

Lastly, having encoding both the decomposition forest g and the deletions and insertions of actions as a SAT formula, we still have to deal with the second hardness source, which is to find a mapping between $yield(g)$ and the given plan π. One could recognize that this bears a lot of similarities to the subgraph isomorphism problem where we can view the plan as one graph and the yield of the decomposition forest as the other, and hence the plan π is a valid linearisation of $yield(g)$ *iff* $yield(g)$ is a subgraph of π. Consequently, we could also employ some SAT encoding of the subgraph isomorphism problem, e.g., the one by Torán (2013), to encode this second hardness source.

More importantly, knowing that ACT$^+$ is the most expensive operation might be a key insight into developing a more sophisticated algorithm. This is because, instead of encoding both action insertions and deletions, we might only need to deal with the insertions, and whether deletions are feasible can then be checked in polynomial time due to Thm. 4.

5 Having White and Black List Plans

We now move on to study the scenario where a domain engineer provides both white list plans and black list plans.

Definition 14. Let $X \subseteq \{$ACT$^+$, ACT$^-$, ORD$^+$, ORD$^-\}$ and $|X| \geq 1$, P be a planning problem, and Π^+ and Π^- be two sets of plans. We define the problem CHANGEWB$_X$ as: Is there a change sequence $\mathbf{\Delta}$ consisting of operations restricted by X such that $P \rightarrow^*_{\mathbf{\Delta}} P'$, and for any $\pi^+ \in \Pi^+$ and $\pi^- \in \Pi^-$, π^+ is a solution to P', and π^- is *not*.

One can easily observe that the problem CHANGE$_X$ is a special case of CHANGEWB$_X$. We thus have the following:

Algorithm 1: An algorithm deciding CHANGEWB$_X$ nondeterministically based on the VERIFYSEQ oracle machine.

```
Non-deterministically choose a change sequence Δ
Apply Δ to P with P →*_Δ P'
for all π⁺ ∈ Π⁺ do
    if VERIFYSEQ(P', π⁺) = false then
        return false
for all π⁻ ∈ Π⁻ do
    if VERIFYSEQ(P', π⁻) = true then
        return false
return true
```

Theorem 6. *Given $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^+, \text{ORD}^-\}$ and $|X| \geq 1$,* CHANGEWB$_X$ *is* $\mathbb{NP}$*-hard.*

In fact, we hypothesize that CHANGEWB$_X$ could fall in the complexity class called Σ_2^p that is beyond $\mathbb{NP}$ under the assumption $\mathbb{P} \neq \mathbb{NP}$. Here, we will present a proof of membership. We first give an introduction to the complexity class Σ_2^p based upon the work by Arora and Barak (2009).

The complexity class Σ_2^p, also denoted as $\mathbb{NP}^{\mathbb{NP}}$, is the set of all decision problems which can be decided in polynomial time by a *nondeterministic* Turning Machine $\mathcal{M}$ with access to an $\mathbb{NP}$ *oracle machine*. An $\mathbb{NP}$ oracle machine is a *hypothesis* machine which can decide any decision problem in $\mathbb{NP}$ in *polynomial* time. In other words, a problem is in Σ_2^p if we can design a *nondeterministic* procedure which decides the problem in polynomial time by invoking a *hypothesis* function which decides a problem in $\mathbb{NP}$ in polynomial time.

Consequently, we will prove Σ_2^p membership of the problem CHANGEWB$_X$ by exploiting the $\mathbb{NP}$ oracle machine which decides the VERIFYSEQ (cf. Def. 8) problem that lies in $\mathbb{NP}$ (Behnke, Höller, and Biundo 2015, Cor. 3) in polynomial time. The nondeterministic algorithm for deciding CHANGEWB$_X$ is shown in Alg. 1.

Theorem 7. *Given an $X \subseteq \{\text{ACT}^+, \text{ACT}^-, \text{ORD}^+, \text{ORD}^-\}$ with $|X| \geq 1$,* CHANGEWB$_X$ *is in* Σ_2^p*.*

Taken together, we know that the problem CHANGEWB$_X$ is within Σ_2^p and at least $\mathbb{NP}$-hard.

6 Discussion

One might observe that we did not consider the operations of adding and deleting methods. The reason for this is twofold. First, the complexity of deciding whether a set of plans can be turned into solutions via adding and deleting methods is still $\mathbb{NP}$-complete, which follows trivially from the results by Lin and Bercher (2021) and which is caused by the third hardness source investigated earlier. Thus, studying these two operations does not provide any novel insight. Second, these two operations can be simulated by those we studied in the paper. Concretely, deleting a method can be simulated by adding an inapplicable action to the method, and adding a method is equivalent to defining an empty template (method) for some tasks and filling them appropriately.

Further, we also did not consider *method preconditions*. Note that there are two semantics of method preconditions, that is, either a method precondition holds *immediately* before the first subtask of the method, or it holds *sometimes* before the first subtask. For the latter one, we can simply compile a method precondition away by introducing a new action at the beginning of this method with the method's precondition as its precondition. For the former, the formalism incorporating such method preconditions does not yet exist.

7 Conclusion

In this paper, we first investigated the computational complexity of deciding whether a set of white list plans can be turned into solutions by modifying an HTN planning domain and clarified every hardness source that makes it $\mathbb{NP}$-hard. Afterwards, we take into account both white list and black list plans and shown that the problem involving both these two sets of plans is at least $\mathbb{NP}$-hard but lies in the class Σ_2^p.

References

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.

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. 2017. This Is a Solution! (... But Is It Though?) – Verifying Solutions of Hierarchical Planning Problems. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling, ICAPS 2017*, 20–28. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT – Totally-Ordered Hierarchical Planning Through SAT. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 6110–6118. AAAI.

Behnke, G.; Höller, D.; and Biundo, S. 2019. Bringing Order to Chaos – A Compact Representation of Partial Order in SAT-Based HTN Planning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, 7520–7529. 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.; Kraus, M.; Schiller, M.; Manstetten, D.; Dambier, M.; Dorna, M.; Minker, W.; Glimm, B.; and Biundo, S. 2021. Do It Yourself, but Not Alone: *Companion*-Technology for Home Improvement – Bringing a Planning-Based Interactive DIY Assistant to Life. *Künstliche Intelligenz – Special Issue on NLP and Semantics*, 35: 367–375.

Cook, S. A. 1971. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, 151–158. ACM.

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

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*, 1955–1961. AAAI.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming Up With Good Excuses: What to do When no Plan Can be Found. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010*, 81–88. AAAI.

Gragera, A.; García-Olaya, Á.; and Fernández, F. 2022. Repair Suggestions for Planning Domains with Missing Actions Effects. In *Proceedings of the 5th International Workshop of Explainable AI Planning, XAIP 2022*.

Greiner, R. 1999. The Complexity of Theory Revision. *Artificial Intelligence*, 107(2): 175–217.

Grover, S.; Sengupta, S.; Chakraborti, T.; Mishra, A. P.; and Kambhampati, S. 2020. RADAR: automated task planning for proactive decision support. *Human-Computer Interaction*, 35(5-6): 387–412.

Helmert, M.; and Lasinger, H. 2010. The Scanalyzer Domain: Greenhouse Logistics as a Planning Problem. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010*, 234–237. AAAI Press.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020. HTN Planning as Heuristic Progression Search. *Journal of Artificial Intelligence Research*, 67: 835–880.

Karpas, E.; and Magazzeni, D. 2020. Automated Planning for Robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3: 417–439.

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*, 4152–4159. IJCAI.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence, AAAI 2023*. AAAI.

Lindsay, A.; Franco, S.; Reba, R.; and McCluskey, T. L. 2020. Refining Process Descriptions from Execution Data in Hybrid Planning Domain Models. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling, ICAPS 2020*, 469–477. AAAI.

Nebel, B.; and Bäckström, C. 1994. On the Computational Complexity of Temporal Projection, Planning, and Plan Validation. *Artificial Intelligence*, 66: 125–160.

Roberts, J. O.; Mastorakis, G.; Lazaruk, B.; Franco, S.; Stokes, A. A.; and Bernardini, S. 2021. vPlanSim: An Open Source Graphical Interface for the Visualisation and Simulation of AI Systems. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling, ICAPS 2021*, 486–490. 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 and the 25th European Conference on Artificial Intelligence, IJCAI-ECAI 2022*, 4657–4664. IJCAI.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2021. Foundations of Explanations as Model Reconciliation. *Artificial Intelligence*, 301: 103558.

Sreedharan, S.; Chakraborti, T.; Muise, C.; Khazaeni, Y.; and Kambhampati, S. 2020. – D3WA+ – A Case Study of XAIP in a Model Acquisition Task for Dialogue Planning. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling, ICAPS 2020*, 488–498. AAAI.

Strobel, V.; and Kirsch, A. 2020. MyPDDL: Tools for Efficiently Creating PDDL Domains and Problems. In *Knowledge Engineering Tools and Techniques for AI Planning*, 67–90. Springer.

Torán, J. 2013. On the Resolution Complexity of Graph Non-isomorphism. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing, SAT 2013*, 52–66. Springer.

Vaquero, T.; Tonaco, R.; Costa, G.; Tonidandel, F.; Silva, J.; and Beck, C. 2012. itSIMPLE4.0: Enhancing the Modeling Experience of Planning Problems. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling, ICAPS 2012*, 11–14.

Viola, C. L.; Orlandini, A.; Umbrico, A.; and Cesta, A. 2019. ROS-TiPlEx: How to make experts in A.I. Planning and Robotics talk together and be happy. In *Proceedings of the 28th IEEE International Conference on Robot and Human Interactive Communication, RO-MAN 2019*, 1–6. IEEE.