# Change the World – How Hard Can that Be? On the Computational Complexity of Fixing Planning Models

**Songtuan Lin** and **Pascal Bercher**
School of Computing, College of Engineering and Computer Science
The Australian National University, Canberra, Australia
{songtuan.lin, pascal.bercher}@anu.edu.au

## Abstract

Incorporating humans into AI planning is an important feature of flexible planning technology. Such human integration allows to incorporate previously unknown constraints, and is also an integral part of automated modeling assistance. As a foundation for integrating user requests, we study the computational complexity of determining the existence of changes to an existing model, such that the resulting model allows for specific user-provided solutions. We are provided with a planning problem modeled either in the classical (non-hierarchical) or hierarchical task network (HTN) planning formalism, as well as with a supposed-to-be solution plan, which is actually not a solution for the current model. Considering changing decomposition methods as well as preconditions and effects of actions, we show that most change requests are NP-complete though some turn out to be tractable.

## 1 Intro

Involving humans in the planning process is an important area of interest within the field of automated planning – in particular when tackling real-world problems, since often not all constraints are known in advance and posed by a human expert during planning [Allen and Ferguson, 2002; Myers *et al.*, 2003]. In particular, the planning model used by a system is usually distinct from what a user expects [Chakraborti *et al.*, 2017; 2020], which may consequently lead to a system not finding any solution at all, or a solution that is rejected by the user. The former can be addressed, e.g., by changing the initial state appropriately [Göbelbecker *et al.*, 2010], whereas the latter, e.g., can be addressed by incorporating change requests on the solution produced [Behnke *et al.*, 2016]. In general, dealing with change requests to plans or models by a human user is formally known as *mixed-initiative planning (MIP)* [Myers *et al.*, 2003]. The scheme of MIP has been successfully applied, e.g., in activities like route planning [Ferguson *et al.*, 1996], the Mars-rovers [Ai-Chang *et al.*, 2004; Bresina *et al.*, 2005], and the evacuation of inhabitants of a fictitious island [Ferguson and Allen, 1998]. Moreover, imposing changes to the model employed by a system is a way to provide *modeling assistance* which aims to help humans identify and fix potential problems in a model. A user could, for example, provide a plan claiming it must be a solution based on the current model. If this is not the case, a (MIP) system could suggest model changes to make it a solution thus finding potential modeling mistakes. Those changes can further be regarded as a *reconciliation* of the current model and the user-desired model, and thus serve as model-based explanations [Chakraborti *et al.*, 2020].

| Methods Given? | Changes | Theorems: *Any Changes* | Theorems: *k Changes* |
|---|---|---|---|
| No | Del | Thm. 1 & 2 | Cor. 1 |
| | Add | Thm. 1 & 3 | |
| | Add, Del | Thm. 1 & 4 | |
| Yes | $*$ | Thm. 5 | Cor. 2 |
| Yes: Unique | $*$ | Thm. 6 | |

Table 1: The investigated problems for changing decomposition methods. "Methods Given" indicates whether a sequence of decomposition methods is given. "Unique" refers to the special case where each method refines a unique task. All classes are NP-complete except the ones listed in the last row, which are in P. The "$*$"s represents all combinations of investigated changes.

| Changes | Complexity | Theorems *k Changes* |
|---|---|---|
| $prec$ | P | Thm. 7 |
| $del$ | | Thm. 8 |
| $prec, del$ | | Thm. 10 |
| $add$ | NP-complete | Thm. 9 |
| $prec, add$ | | Thm. 11 |
| $del, add$ | | Thm. 12 |
| $prec, add, del$ | | Thm. 13 |

Table 2: The investigated problems, complexity results, and theorems for changing preconditions and effects.

We assume that we are given a sequence of actions that is supposed to be a solution, but it is actually not: either because it is not executable, or because it does not fulfill additional constraints posed on desired solutions. For the latter, we consider the framework of Hierarchical Task Network (HTN) planning (see the work by Bercher *et al.* [2019] for a recent survey), which is a hierarchical approach to planning where a set of initially given abstract activities need to be refined until a primitive (executable) action sequence was obtained. Relying on this approach allows more control on the produced plans than classical planning as only plans are allowed that follow the user-specified hierarchy [Höller *et al.*, 2014; 2016], and was thus also studied and applied in MIP before [Myers *et al.*, 2003; Bresina *et al.*, 2005; Behnke *et al.*, 2016]. To make the action sequence in question a solution to the problem, we consider changing the actions' preconditions or effects (the obtained results are thus relevant to both hierarchical and non-hierarchical planning), as well as changing the model's rules on how plans can be obtained (so-called decomposition methods) by either adding or removing actions to them.

The objective of this paper is to establish the computational complexity of checking the feasibility of such model changes. In HTN planning, checking for feasible changes to the current solution range between NP and undecidability [Behnke *et al.*, 2016], similar to deciding HTN problems [Erol *et al.*, 1996; Geier and Bercher, 2011; Alford *et al.*, 2015]. It turns out that model changes are much easier, namely at most NP-complete.

For changing the task hierarchy, we will restrict ourselves to totally-ordered HTN planning, which is the simplest version of HTN planning and also plays a pivot role in practice as seen by a vast majority of totally ordered planning benchmarks at the International Planning Competition 2020 on HTN planning as well as by the increasing body of research dedicated to total-order HTN planning [Marthi *et al.*, 2007; Alford *et al.*, 2009; Behnke *et al.*, 2018; Schreiber *et al.*, 2019; Behnke and Speck, 2021; Olz *et al.*, 2021]. Furthermore all complexity results obtained will directly serve as lower bounds for the partially ordered setting. Changing preconditions and effects is agnostic against the hierarchy. An overview of our complexity results is given in Tables 1 and 2.

## 2 HTN Planning

We first introduce the deployed HTN planning formalism, which is a combination of the one by Bercher *et al.* [2019] and Behnke *et al.* [2018], the latter introducing a simplification for the total order setting.

**Definition 1** (Planning Problem)**.** A (totally ordered) HTN planning problem $P$ is a tuple $(D, tn_I, s_I, g)$ where 1) $D = (F, N_p, N_c, \delta, M)$ is called the domain of $P$, in which $F$ is a finite set of facts, $N_p$ is a finite set of primitive task names, $N_c$ is a finite set of compound task names with $N_c \cap N_p = \emptyset$, $\delta : N_p \to 2^F \times 2^F \times 2^F$ is a function that maps primitive task names to their actions, and $M \subseteq N_c \times (N_p \cup N_c)^*$ is a set of (decomposition) methods. 2) $tn_I \subseteq (N_p \cup N_c)^*$ is the initial task network. 3) $s_I \in 2^F$ is the initial state. 4) $g \subseteq F$ is the goal description.

In the following we will skip the "totally ordered", since we will restrict to this setting throughout the paper. At the centre of the HTN planning formalism is the concept of the task network, which in the total order setting is simply a sequence of task names. Such names are either 1) A primitive task name $p \in N_p$, which is mapped to its action through the function $\delta$. The action of $p$, $\delta(p) = (prec, add, del) \in 2^F \times 2^F \times 2^F$, consists of $p$'s precondition, add, and delete list, respectively. If $\delta(p) = (prec, add, del)$, we also write $(prec(p), add(p), del(p))$ for short. 2) A compound task name $c \in N_c$, which can be refined (decomposed) into a task network $tn = t_1 \ldots t_n$ by applying a method $m = (c, tn) \in M$, where $tn$ is a task network. We write $tn(m)$ to refer to $tn$ of $m$, and $|tn|$ to refer to the length of $tn$.

**Definition 2** (Decomposition)**.** Let $tn = tn_1\, c\, tn_2$ be a task network where $tn_1$ and $tn_2$ are two sequences of task names, and $c \in N_c$ is a compound task name, and $m = (c, tn_m) \in M$ be a method. We say $m$ decomposes $tn$ into another task network $tn'$, written $tn \to_m tn'$, such that $tn' = tn_1\, tn_m\, tn_2$.

Similarly, given a sequence of methods $\overline{m} = m_1 \cdots m_n$, a task network $tn$ is decomposed into another task network $tn'$ by applying $\overline{m}$, written $tn \to^*_{\overline{m}} tn'$, if and only if there exists a sequence of task networks $tn_0 \cdots tn_n$ 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 (primitive) task network $tn = p_1 \cdots p_n$ (also called an action sequence) is said to be executable in a state $s \in 2^F$ if and only if for each $1 \leq i \leq n$, $p_i$ is a primitive task name, and there exists a sequence of states $s_0 \cdots s_n$ such that $s_0 = s$ and for each $1 \leq j \leq n$, $prec(p_j) \subseteq s_{j-1}$ and $s_j = (s_{j-1} \backslash del(p_j)) \cup add(p_j)$. The state $s_n$ is the state produced by $tn$.

**Definition 3** (Solution)**.** Let $P = (D, tn_I, s_I, g)$ be an HTN planning problem. A task network $tn$ is a solution (or plan) to $P$ if and only if $tn$ is executable in $s_I$, it generates a state $s' \supseteq g$, and there exists a sequence of methods $\overline{m}$ that refines $tn_I$ into $tn$, i.e., $tn_I \to^*_{\overline{m}} tn$.

We will now start our investigations where we have a primitive task network $tn$ given (i.e., an action sequence), which is not a solution – but *should* be according to a user. According to the solution criteria there are just two possible reasons: Either $tn$ is not executable/does not satisfy all goals[1], or it does but can not be obtained from the available decomposition methods. To make $tn$ a solution we first consider changing the available methods of the model in Section 3 and then consider changing the action definitions in Section 4.

## 3 Correcting the Model: Changing Methods

Given an action sequence that cannot be obtained by a sequence of decomposition methods, we want to change the model so that it can. For this, we first need to consider the allowed changes, which will be adding and deleting actions from the decomposition methods – they will be formally defined next. We start with the ADD-TASK operation, which specifies at which position in a method's task sequence a given action may be added.

**Definition 4** (ADD-TASK)**.** Let $p$ be a primitive task name, $m = (c, tn)$ with $tn = t_1 \cdots t_n$ be a method, and $1 \leq i \leq n + 1$ be an integer. The operation ADD-TASK is a function that takes as inputs $m$, $p$, and $i$ and outputs a new method

[1] We do not differentiate between an action sequence that is not executable and a sequence that is executable but does not satisfy all goals, since the latter can be regarded a special case of the former.

$m' = (c, tn')$, written $m' = \text{ADD-TASK}(m, p, i)$, such that $tn' = tn_1\, p\, tn_2$ where $tn_1 = t_1 \cdots t_{i-1}$ and $tn_2 = t_i \cdots t_n$.

The operation DEL-TASK, which allows the primitive task name in a specific position to be removed from a method, is the analogous operation to ADD-TASK for action deletion.

**Definition 5** (DEL-TASK)**.** Let $m = (c, tn)$ be a method where $tn = tn_1\, p\, tn_2$ with $tn_1 = t_1 \cdots t_{i-1}$ and $tn_2 = t_{i+1} \cdots t_n$ be two sequences of task names, and $p$ be a primitive task name. The operation DEL-TASK is a function that takes as inputs $m$ and $i$ and outputs a new method $m' = (c, tn')$, written $m' = \text{DEL-TASK}(m, i)$, such that $tn' = tn_1\, tn_2$.

Given two methods $m$ and $m'$ and a sequence of method-change operations $\mathcal{X} = x_1(m_1, *) \cdots x_n(m_n, *)$ where for each $1 \leq i \leq n$, $x_i$ is either ADD-TASK or DEL-TASK, $m_i$ is a method, and $*$ refers to the remaining parameters in the operation $x_i$, we write $m \to^*_{\mathcal{X}} m'$ if $m_1 = m$, $m' = x_n(m_n, *)$, and $m_{i+1} = x_i(m_i, *)$ for each $1 \leq i \leq n-1$. Informally, $m \to^*_{\mathcal{X}} m'$ indicates that the method $m'$ can be obtained from $m$ by applying a sequence of method-change operations $\mathcal{X}$. We use $|\mathcal{X}|$ to refer to the length of $\mathcal{X}$.

**Definition 6** (Model Change)**.** Let $P = (D, tn_I, s_I, g)$ with $D = (F, N_p, N_c, \delta, M)$ and $M = \{m_1, \cdots, m_n\}$ be a planning problem, and $\mathcal{X}$ be a sequence of method-changes. A problem $P' = (D', tn_I, s_I, g)$ with $D' = (F, N_p, N_c, \delta, M')$ and $M' = \{m'_1, \cdots, m'_n\}$ is obtained from $P$ by applying $\mathcal{X}$, written $P \to^*_{\mathcal{X}} P'$, if and only if for each $1 \leq i \leq n$, either $m'_i = m_i$ or there exists a subsequence[2] $X_i$ of $\mathcal{X}$ such that $m_i \to^*_{X_i} m'_i$.

The definition above highlights that after applying $\mathcal{X}$, nothing changes except the methods in $P$, which maintain a one-to-one mapping to that in $P'$. For convenience, we use $\beta_{\mathcal{X}} : M \to M'$ to denote this mapping, where for each method $m_i$ with $1 \leq i \leq n$, $\beta_{\mathcal{X}}(m_i) = m'_i$.

Now that we have defined all necessary changes, we can move on to investigate the computational complexity of checking whether such changes exist that turn the given plan into a solution. We will consider two cases, where either we are only given the action sequence, or we are also given a method sequence that is supposed to generate the task network in question – but didn't. We start by investigating the former case.

### 3.1 Complexity of Fixing the Methods – Given Just an Action Sequence

We first define the decision problem of fixing the method set by an arbitrary number of add or delete operations.

**Definition 7** (FIX-METHS$_X$)**.** Let $P$ be a planning problem, and $tn$ be a task network. We define the decision problems FIX-METHS$_X$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$ as: Is there a sequence of method-change operations $\mathcal{X}$, such that $P \to^*_{\mathcal{X}} P'$, $tn$ is a solution to $P'$, and $\mathcal{X}$ consists of ADD-TASK and DEL-TASK operations, according to $X$?

[2] We define a subsequence in a conventional way where a sequence $\overline{a}'$ is said to be a subsequence of another sequence $\overline{a}$ if $\overline{a}'$ can be obtained from $\overline{a}$ by removing some elements from it.

One immediate observation is that if there exists some change sequence that turns $tn$ into a solution, there must be one of length bounded by a polynomial.

**Lemma 1.** *Let $P$ and $tn$ be the planning problem and the task network given by an instance of* FIX-METHS$_X$ *with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$. If there exists some sequence of method-change operations that turns $tn$ into a solution, then there exists a sequence $\mathcal{X}$ such that $P \to^*_{\mathcal{X}} P'$, $tn$ is a solution to $P'$, and $|\mathcal{X}| \leq |tn| + \sum_{m \in M} |tn(m)|$.*

*Proof.* We only consider the case where $X = \{\text{ADD}, \text{DEL}\}$ because the upper bound of the sequence $\mathcal{X}$ in question in this case is strictly larger than that in the remaining ones.

To prove the argument, we only need to show that the shortest change sequence that turns $tn$ into a solution has the upper bound in question. Suppose $\mathcal{X}$ is the shortest change sequence that turns $P$ into another planning problem $P'$ such that $tn$ is a solution to $P'$. $\mathcal{X}$ can contain at most $\sum_{m \in M} |tn(m)|$ deletions, which is obtained by removing all actions from all methods. Similarly, $\mathcal{X}$ contains at most $|tn|$ additions because the number of additions cannot exceed the size of $tn$. Thus, $|\mathcal{X}| \leq |tn| + \sum_{m \in M} |tn(m)|$. ☐

Membership of the problem under investigation can thus be easily determined by this lemma.

**Theorem 1.** FIX-METHS$_X$ *with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$ is in NP.*

*Proof.* For each variant, we can always guess a change sequence $\mathcal{X}$ whose length is bounded by a polynomial according to Lem. 1. Transforming $P$ into $P'$ thus takes polynomial time. Verifying whether $tn$ is a solution to $P'$ can also be done in polynomial time by regarding $P'$ as a context-free grammar and $tn$ as a string [Behnke *et al.*, 2015]. ☐

We now investigate the hardness of these problems.

**Theorem 2.** FIX-METHS$_{\text{DEL}}$ *is NP-hard.*

*Proof.* We reduce from the *independent set* problem, which is known to be NP-complete [Korte and Vygen, 2008]. Let $k, n, r \in \mathbb{N}$, and $G = (V, E)$ with $V = \{v_1, \cdots v_n\}$ and $E = \{e_1, \cdots, e_r\}$ where $e_i = (v, v')$ with some $v, v' \in V$ for each $1 \leq i \leq r$ be a graph. A solution to an *independent set* problem instance is a subset $S$ of $V$ such that $|S| = k$, and no two vertices in $S$ are adjacent, i.e., no two vertices are connected by an edge.

To construct an equivalent FIX-METHS$_{\text{DEL}}$ instance, we first construct a planning problem $P = (D, tn_I, s_I, g)$ with $D = (F, N_c, N_p, \delta, M)$ as follows. We let $F = \emptyset$, $s_I = \emptyset$, $g = \emptyset$, and $\delta : N_p \to \{(\emptyset, \emptyset, \emptyset)\}$. For each vertex $v_i$ (with $1 \leq i \leq n$) in $V$, we construct a compound task $v^c_i$. For each edge $e_i$ ($1 \leq i \leq r$) in $E$, we construct one primitive task $e^p_i$ and one compound task $h^c_i$ ($h^c_i$ is used as a placeholder, which will be explained shortly). Additionally, we construct one more primitive task $s$ (which stands for '*selected*'). Taken together, $N_c = \{v^c_1, \cdots, v^c_n, h^c_1, \cdots, h^c_r\}$ and $N_p = \{e^p_1, \cdots, e^p_r, s\}$. Afterwards, for each compound task $v^c_i$ (with $1 \leq i \leq n$), we construct a method $m^s_i = (v^c_i, s)$ which stands for selecting vertex $v_i$ into $S$. For each $h^c_i$ (with

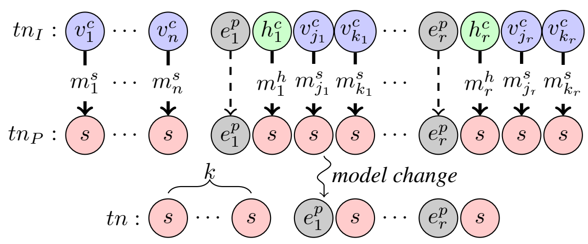


Figure 1: The constructions of $tn_I$ and $tn$. $tn_I$ is decomposed into $tn_P$ by a sequence of methods, and changing the model by emptying the appropriate methods will turn $tn_P$ into $tn$.

$1 \le i \le r$), we construct a method $m_i^h = (h_i^c, s)$. Thus, $M = \{m_1^s, \cdots, m_n^s, m_1^h, \cdots, m_r^h\}$. We then construct $tn_I$ to encode the structure of $G$, as shown in Figure 1.

The prefix sequence $v_1^c \cdots v_n^c$ encodes all the vertices in $G$, and each subsequence $e_i^p\, h_i^c\, v_{j_i}^c\, v_{k_i}^c$ with $1 \le i \le r$ and $1 \le j_i, k_i \le n$ indicates that the two endpoints of the edge $e_i$ are $v_{j_i}$ and $v_{k_i}$. Any method sequence which decomposes $tn_I$ into a solution $tn_P$ should be a permutation of the one shown in Figure 1, and it encodes how the vertices are selected into $S$. At the beginning, all vertices are selected. Thus, applying a DEL-TASK operation to a method $m_i^s$ ($1 \le i \le n$) is analogous to deselecting vertex $v_i$ from the set $S$.

Lastly, the solution to the *independent set* problem instance is encoded using the plan $tn$ which is shown in Figure 1. The prefix sequence which repeats $s$ $k$ times indicates that there should be $k$ selected vertices. The suffix sequence indicates that for each edge $e_i$ (with $1 \le i \le r$), *at most* one of its endpoints can be selected. Moreover, a solution to the *independent set* instance may result in a situation where both endpoints of some edge are not selected. The placeholders $h_1^c, \cdots, h_r^c$ in $tn_I$ are used to encounter this situation. For some edge $e_i$ ($1 \le i \le r$) 1) if a solution to the *independent set* instance asserts that only one endpoint of $e_i$ should be deselected, the respective DEL-TASK operation will be applied to $m_i^h$, which results in the subsequence $e_i^p\ s$ to match $tn$, otherwise 2) $m_i^h$ will not be modified, which also results in the subsequence $e_i^p\ s$.

It then follows that the given *independent set* instance has a *yes* answer if and only if the FIX-METHS$_{\text{DEL}}$ instance we constructed has one. Further, observe that the size of $N_p$, $N_c$, and $M$ are bounded by $r+1$, $n+r$, and $n+r$, respectively, and $|tn_I|$ is bounded by $n+4r$, the planning problem can be constructed in time $\mathcal{O}(n+r)$. Additionally, since $|tn| = k + 2r \le n + 2r$ (because $k$ cannot exceed the number of vertices in the graph), we can conclude that the reduction can be done in polynomial time. □

We now consider the hardness when only additions are allowed. Since this is somehow analogous to the deletion case, we non-surprisingly obtain NP-hardness as well:

**Theorem 3.** FIX-METHS$_{\text{ADD}}$ *is NP-hard.*

*Proof.* We again reduce from the *independent set* problem. We construct a plan $tn$ and a planning problem $P$ which are identical to those we constructed in the proof of Thm. 2 except that for each $1 \le i \le n$, $m_i^s = (v_i^c, \varepsilon)$, and for each $1 \le j \le r$, $m_j^h = (h_j^c, \varepsilon)$, where $\varepsilon$ denotes an empty task network. One can easily verify that there exists solely one primitive task network (solution) into which $tn_I$ can be refined, i.e., $e_1^p \cdots e_r^p$. Applying ADD-TASK to a method $m_i^s$ is now analogous to selecting the vertex $v_i$ into the set $S$, and we can always apply ADD-TASK to some $m_j^h$ with $1 \le j \le r$ in the case where none of the endpoints of $e_j$ are selected. It follows that the *independent set* problem has a *yes* answer if and only if the problem we constructed has one. □

We now show the computational hardness of the general case, where all change operations can be used.

**Theorem 4.** FIX-METHS$_{\text{ADD,DEL}}$ *is NP-hard.*

*Proof.* In the proof of Thm. 3 we let each $m_i^s$ with $1 \le i \le n$ and each $m_j^h$ with $1 \le j \le r$ in the constructed planning problem contain an empty task network, thus making the DEL-TASK operation redundant (or pointless). Thus, by applying the same reduction, we directly obtain hardness. □

So far, we only asked whether *any* number of changes exists that makes the given task network a solution. We now check whether the problem becomes harder when we are interested in the *minimal* number of required changes. We formalize this in terms of limiting the size $k$ of the method-change operation sequence.

**Definition 8** (FIX-METHS$_X^k$)**.** Let $k \in \mathbb{N}$. For $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \ge 1$, the problems FIX-METHS$_X^k$ are identical to FIX-METHS$_X$, except that we demand that any sequence of model changes is limited in size by $k$, i.e., $|\mathcal{X}| \le k$.

These problems, apparently, do not become harder than the original, unbounded ones.

**Corollary 1.** FIX-METHS$_X^k$ *with* $X \subseteq \{\text{ADD}, \text{DEL}\}$ *and* $|X| \ge 1$ *is NP-complete.*

*Proof. Membership:* Let $L = \sum_{m \in M} |tn(m)|$. For each FIX-METHS$_X^k$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \ge 1$, we have already shown in Lem. 1 that each shortest change sequence that turns $tn$ into a solution must be limited by a polynomial. Thereby, although the given $k$ can be exponentially large via encoding it logarithmically, a change sequence does never have to be of length of that worst-case exponential value. More precisely, it should be of length smaller or equal to the minimum of the requested number of $k$ changes and the polynomial bound $L + |tn|$. Guessing such a sequence $\mathcal{X}$, it takes polynomial time to change $P$ to $P'$ [Behnke *et al.*, 2015]. Further, since deciding whether $tn$ is a solution to $P'$ takes polynomial time, the total time required to verify whether $\mathcal{X}$ is a correct sequence is also a polynomial.

*Hardness:* For each FIX-METHS$_X^k$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \ge 1$, we reduce from FIX-METHS$_X$. Given an instance of FIX-METHS$_X$, since a shortest change sequence for it cannot have length greater than $L + |tn|$ (with $L = \sum_{m \in M} |tn(m)|$), we can construct a FIX-METHS$_X^k$ instance which is identical to the FIX-METHS$_X$ instance and has $k = L + |tn|$. Hardness then follows directly. □

### 3.2 Complexity of Fixing the Methods – Given an Action and Method Sequence

We move on to consider the computational hardness of problems where we are given both an action sequence *and* a decomposition method sequence that is supposed to generate it. This eliminates one possible source of computational hardness, because there won't be a choice which method to choose per compound task. Though one practical motivation is again failed plan verification. Suppose a flawed HTN planning system that claims that a plan can be produced based on a sequence of used methods, whereas an independent plan verification system [Behnke *et al.*, 2017; Barták *et al.*, 2018; 2020] tells that this is not the case. Checking which methods are flawed can then serve as counter-factual explanation [Ginsberg, 1986; Chakraborti *et al.*, 2017; 2020] to point towards implementation errors (showing the user which methods were not correctly processed by the planning system).

We start with the decision problems which allow an arbitrary number of changes.

**Definition 9** (FIX-SEQS$_X$)**.** Let $P$ be a planning problem, $\overline{m} = m_1 \cdots m_n$ be a sequence of methods, and $tn$ be a task network. We define the decision problems FIX-SEQS$_X$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$ as: Is there a sequence of method-change operations $\mathcal{X}$ such that $P \rightarrow^*_{\mathcal{X}} P'$, $tn_I \rightarrow^*_{\overline{m'}} tn$ with $\overline{m'} = \beta_{\mathcal{X}}(m_1) \cdots \beta_{\mathcal{X}}(m_n)$, and $\mathcal{X}$ consists of only DEL-TASK and ADD-TASK operations according to $X$?

As before all variants turn out to be equally hard:

**Theorem 5.** *FIX-SEQS$_X$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$ is NP-complete.*

*Proof. Membership:* Let $L = \sum_{i=1}^{n} |tn(m_i)|$. For each FIX-SEQS$_X$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$, one immediate observation is that if there exists some change sequence that turns $tn$ into a solution, there must be one (assuming all operations refer to methods in $\overline{m}$) whose length is bounded by $L + |tn|$, which contains at most $L$ deletions and $|tn|$ additions. Guessing such a sequence, transforming $P$ into $P'$ thus takes time $\mathcal{O}(L + |tn|)$, and it follows that $L' = \sum_{i=1}^{n} |tn(\beta_{\mathcal{X}}(m_i))| \leq L + |tn|$. Thereby, checking whether $\overline{m'}$ decomposes $tn_I$ to $tn$ can be done in time $\mathcal{O}(L') = \mathcal{O}(L + |tn|)$. Taken together, verifying whether $\mathcal{X}$ is a correct operation sequence takes P-time.

*Hardness:* For each FIX-SEQS$_X$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$, we again reduce from the *independent set* problem. We construct a plan $tn$ and a planning problem $P$ which are identical to those in the hardness proof of the corresponding FIX-METHS$_X$ problem. There, we have justified that any method sequence that refines $tn_I$ into a solution is a permutation of the one shown in Figure 1. Thus, by explicitly constructing such a method sequence, we complete the reduction. Hardness then follows directly. □

We now formalize the problems which we use to formalize finding an *optimal* (i.e., shortest) sequence of changes.

**Definition 10** (FIX-SEQS$^k_X$)**.** Let $k \in \mathbb{N}$. For each $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$, the problem FIX-SEQS$^k_X$ is identical to FIX-SEQS$_X$ except that we demand that any sequence of change operations is limited in size by $k$, i.e., $|\mathcal{X}| \leq k$.

Following the idea used in the proof of Cor. 1, we immediately obtain the following result.

**Corollary 2.** *The problems FIX-SEQS$^k_X$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$ are NP-complete.*

Under certain conditions the problem becomes tractable.

**Theorem 6.** *FIX-SEQS$^k_X$ and FIX-SEQS$_X$ with $X \subseteq \{\text{ADD}, \text{DEL}\}$ and $|X| \geq 1$ can be decided in polynomial time if $tn_I$ contains no primitive tasks, and each method in the method sequence refines a unique task, i.e., for each $1 \leq i, j \leq n$ with $i \neq j$, if $m_i = (c_i, tn_i)$ and $m_j = (c_j, tn_j)$, then $c_i \neq c_j$.*

*Proof.* We only show the proof for the case where both additions and deletions are allowed, but the same idea can be applied to the other cases. The idea is to compare $k$ with the minimal number of changes required. To find that number, we first apply the method sequence $\overline{m}$ to $tn_I$, which results in a solution $tn_P$. We can regard $tn_P$ and $tn$ as strings where each task is a symbol. Since each method refines a unique task, changes that are imposed to one method will not have an impact on another. In other words, we can directly apply additions and deletions to $tn_P$ without considering the method sequence. Thus, the problem of finding the minimal number of changes required to transform $P$ into $P'$ such that $tn_I \rightarrow^*_{\overline{m'}} tn$ can then be reduced to finding the minimal number of editions required to transform $tn_P$ into $tn$. That is equivalent to the *string edit distance* problem [Masek and Paterson, 1980], which takes as inputs two strings $S$ and $S'$, and outputs an edit sequence $E$ that consists of two types of editions: adding a character and deleting a character, and changes $S$ to $S'$ in the minimal number of steps. In our case, the two strings are $tn_P$ and $tn$, and the length of $E$ is the minimal number of changes required. Since the *string edit distance* problem is solvable in polytime, and comparing the minimal number of changes with $k$ takes polynomial time as well, the $k$-bounded problem can be solved in P. Clearly, the unbounded case can thus also be decided in P. □

## 4 Correcting the Model: Changing Tasks

We now investigate the case where we are given an action sequence that is not executable, and we aim to make it executable by changing its actions' preconditions and effects. So here we ignore the hierarchy completely, either because a non-hierarchical problem was solved in the first place, or because the plan is already proved to be obtainable by the available methods. Note that actions may occur multiple times, so changing one action results in changing all the other identical actions in the same sequence as well. We begin our investigation by introducing the allowed changes. One can easily verify that a reasonable change made to an action $p$ can be categorized as being: 1) Removing a fact from $prec(p)$. 2) Removing a fact from $del(p)$. 3) Adding a fact to $add(p)$.

We formalize these changes as follows.

**Definition 11** (FIX-PREC)**.** Let $p$ be an action and $f \in prec(p)$ be a fact. The operation FIX-PREC is a function that takes as inputs $p$ and $f$, and outputs new preconditions of $p$ such that $\delta(p) = (prec(p) \backslash \{f\}, add(p), del(p))$.

**Definition 12** (FIX-ADD)**.** Let $p$ be an action and $f \in F$ be a fact. The operation FIX-ADD is a function that takes as inputs $p$ and $f$, and outputs new effects of $p$ such that $\delta(p) = (prec(p), add(p) \cup \{f\}, del(p))$.

**Definition 13** (FIX-DEL)**.** Let $p$ be an action and $f \in del(p)$ be a fact. The operation FIX-DEL is a function that takes as inputs $p$ and $f$, and outputs new effects of $a$ such that $\delta(p) = (prec(p), add(p), del(p) \backslash \{f\})$.

Having defined the allowed operations, we now proceed to examine the complexity of making an action sequence executable. Notice that in most cases, deciding whether *any* such modifications *exist* is rather trivial and can be done in polynomial time or even constant time. We will thus focus on checking whether $k$ changes are sufficient.

**Definition 14** (FIX-ACTIONS$^k_X$)**.** Let $k$ be an integer, $tn$ be an action sequence. We define the problems FIX-ACTIONS$^k_X$ with $X \subseteq \{\text{PREC}, \text{ADD}, \text{DEL}\}$ and $|X| \geq 1$ as follows: Is it possible to make $tn$ executable by applying the allowed FIX operation(s) (Defs. 11 to 13) according to $X$ at most $k$ times?

We start with the simplest questions where we are only allowed to use one of the three operations.

**Theorem 7.** FIX-ACTIONS$^k_{\text{PREC}}$ *is in P.*

*Proof.* Given an action sequence $tn = p_1 \cdots p_n$, we compare $k$ with the minimal number of changes required to make $tn$ executable, which can be found as follows. For any action $p_i$ with $1 \leq i \leq n$, if there exists some fact $f \in prec(p_i)$ that cannot be satisfied, we apply FIX-PREC to remove it. One can easily verify that the total number of FIX-PREC applied is minimal. Thus, the answer can be obtained by comparing that number with $k$, and the total time required is $\mathcal{O}(|tn|)$. □

**Theorem 8.** FIX-ACTIONS$^k_{\text{DEL}}$ *is in P.*

*Proof.* Let $tn = p_1 \cdots p_n$ be the given action sequence. We again compare $k$ with the minimal number of changes required to make $tn$ executable. For each $p_i$ with $1 \leq i \leq n$ that contains some fact $f \in prec(p_i)$ which is not satisfied in the current state, we find *all* actions $p_j$ with $j < i$, $f \in del(p_j)$, and $f \notin add(p_r)$ for each $j < r < i$, and apply FIX-DEL to remove $f$ from $del(p_j)$. If no such action can be found, or $f$ is not satisfied after all $p_j$s have been processed, it is not possible to make $tn$ executable. For every such pair of $p_j$ and $p_i$, if $f$ remains in $del(p_j)$, the precondition of $p_i$ cannot be satisfied. Thus, we can conclude that the number of FIX-DEL operations applied is minimal, and comparing it with $k$ takes polynomial time. Thus, FIX-ACTIONS$^k_{\text{DEL}}$ is in P. □

Unlike the previous ones, the problem becomes harder when we are only allowed to use FIX-ADD.

**Theorem 9.** FIX-ACTIONS$^k_{\text{ADD}}$ *is NP-complete.*

*Proof. Membership:* Let $tn = p_1 \cdots p_n$ be the given action sequence, and $L = \sum_{i=1}^{n} |prec(p_i)|$. We can bound the number of changes that make $tn$ executable by $L|tn|$, which we get by adding *all* facts to *all* actions in $tn$. Thus, even though the given $k$ can be exponentially large via encoding it logarithmically, there always exists a way to change actions in polynomial many steps if one exists at all. Guessing at most $\min\{k, L|tn|\}$ changes, it takes time $\mathcal{O}(\min\{k, L|tn|\})$ to change the action sequence and $\mathcal{O}(|tn|)$ to verify whether the changed sequence is executable.

*Hardness:* We reduce from the *set covering* problem, which is known to be NP-complete [Karp, 1972]. Let $\tau$ and $\mathcal{S} = \{S_1, \cdots, S_m\}$ be the integer and the set of sets given by an instance of the *set covering* problem, respectively. The solution to this problem instance is a subset $\mathcal{T} \subseteq \mathcal{S}$ such that $|\mathcal{T}| \leq \tau$ and $\bigcup_{T \in \mathcal{T}} T = \bigcup_{i=1}^{m} S_i$. We use the notation $\mathcal{U} = \bigcup_{i=1}^{m} S_i$ to refer to the universal set. Without loss of generality, suppose $\mathcal{U} = \{e_1, \cdots, e_n\}$. We construct an equivalent instance of FIX-ACTIONS$^k_{\text{ADD}}$ as follows. Let $f$ be a dummy fact, for each $e_i \in \mathcal{U}$ with $1 \leq i \leq n$, we construct an action $p_i$ such that $prec(p_i) = \{f\}$, $add(p_i) = \emptyset$, and $del(p_i) = \{f\}$, and we place these actions sequentially: $tn = p_1 \cdots p_n$. Afterwards, for each $S_j \in \mathcal{S}$ with $1 \leq j \leq m$, we construct an action $a_j$ with $prec(a_j) = \emptyset$, $add(a_j) = \emptyset$, and $del(a_j) = \emptyset$. For each $e_i \in \mathcal{U}$ with $1 \leq i \leq n$, if $e_i \in S_j$, we insert $a_j$ into a position between $t_{i-1}$ and $t_i$ in $tn$. Particularly, if $i - 1 = 0$, $a_j$ is inserted before $p_1$. For example, if a set $S_j = \{e_1, e_3, e_4\}$, the action $a_j$ will be inserted to the positions shown: $a_j p_1 p_2 a_j p_3 a_j p_4 \cdots p_n$

Lastly, let $k = \tau$. By construction, if we apply FIX-ADD to some action $a_r$ ($1 \leq r \leq m$), it will resolve the flaws between $a_r$ and the actions that occur after it and have fact $f$ as a precondition. That is equivalent to select $S_r$ into $\mathcal{T}$, and vice versa. Additionally, although applying FIX-ADD to some $p_i$ ($1 \leq i \leq n$) is possible, it is useless because each $p_i$ is preceded by at least one $a_j$ ($1 \leq j \leq m$). Thus, the instance of the *set covering* problem has a *yes* answer if and only if the instance we constructed has one. □

We now start to examine whether the problems become harder when multiple operations are involved.

**Theorem 10.** FIX-ACTIONS$^k_{\text{PREC,DEL}}$ *is in P.*

*Proof.* The idea is again to compare $k$ with the minimal number of changes required to make $tn$ executable. Observe that all facts that occur in the action sequence and may result in flaws are independent of each other. We can consequently deal with them one after another. Let $tn = p_1 \cdots p_n$ be the action sequence, $\mathcal{F} = \bigcup_{i=1}^{n} F_i$ with $F_i = prec(p_i) \cup add(p_i) \cup del(p_i)$ be the set of all facts involved in $tn$. For each $f \in \mathcal{F}$, we do the following:

(1) For each action $p_i$ with $1 \leq i \leq n$ in $tn$, if $f \notin F_i$, we remove $p_i$ from $tn$. Without loss of generality, we denote the new action sequence after this step as $tn' = p'_1 \cdots p'_r$ ($r \leq n$). If $tn'$ is executable, we move on to process the next fact in $\mathcal{F}$, otherwise, we continue to the next step.

(2) For each action $p'_i$ with $1 \leq i \leq r$ in $tn'$, if $f \in prec(p'_i) \cap del(p'_i)$, we split $p'_i$ into two consecutive actions $p'_{i,\top}$ and $p'_{i,\bot}$ such that $prec(p'_{i,\top}) = prec(p'_i)$, $add(p'_{i,\top}) = \emptyset$, $del(p'_{i,\top}) = \emptyset$, $prec(p'_{i,\bot}) = \emptyset$, $add(p'_{i,\bot}) = add(p'_i)$, and $del(p'_{i,\bot}) = del(p'_i)$. We denote the new action sequence after this step as $tn^* = p^*_1 \cdots p^*_k$ ($r \leq k$).

(3) We construct an undirected graph $G = (V, E)$ where $V = \{p^*_1, \cdots, p^*_k\}$, and for any two nodes $v, v' \in V$,

$(v, v') \in E$ if and only if there exist $i, j$ with $i < j$ such that $v = p_i^*$, $v' = p_j^*$, $f \in del(p_i^*)$, $f \in prec(p_j^*)$, for each $l$ with $i \leq l < j$, $f \notin add(p_l^*)$, and there exist an integer $q$ with $1 \leq q < i$ and $f \in add(p_q^*)$.

An edge $(v, v') \in E$ asserts that there exist actions $v$ and $v'$ in $tn^*$ such that $v$ deletes the fact $f$ that is required by $v'$, which consequently make $tn^*$ nonexecutable. Thus, for each such edge $(v, v') \in E$, we should remove $f$ from either 1) $prec(v')$, 2) $del(v)$, 3) or both. For finding the minimal number of changes $N_f$ required to fix the flaws associated with $f$, one can immediately note that this is equivalent to finding the minimum vertex cover of $G$, which can be solved in polynomial time when $G$ is a bipartite graph [Korte and Vygen, 2008]. Thus, the minimal number of changes required to make $tn$ executable can be obtained by summing up $N_f$ for each $f \in \mathcal{F}$. Since $|\mathcal{F}|$ is bounded by a polynomial, and calculating $N_f$ for each $f$ also takes polynomial time, the minimal number of changes required can be obtained in polynomial time. Further, comparing this number with $k$ takes polynomial time. The problem is thus in P. □

**Theorem 11.** FIX-ACTIONS$^k_{\text{PREC,ADD}}$ *is NP-complete.*

*Proof. Membership:* Let $tn = p_1 \cdots p_n$ be the given action sequence, and $\mathcal{F} = \bigcup_{i=1}^n prec(p_i) \cup add(p_i) \cup del(p_i)$. At most $2|\mathcal{F}||tn|$ changes are required if $tn$ can be made executable, which is the number obtained by adding *all* facts to *all* actions and removing *all* facts from the preconditions of *all* actions in $tn$. After guessing a sequence of changes which is of length smaller or equal to the minimum of $k$ and $2|\mathcal{F}||tn|$ it takes $\mathcal{O}(\min\{k, 2|\mathcal{F}||tn|\})$ time to change the action sequence and $\mathcal{O}(|tn|)$ to verify executability.

*Hardness:* We reduce from the FIX-ACTIONS$^k_{\text{ADD}}$ problem. Let $k$ and $tn = p_1 \cdots p_n$ be the integer and the action sequence given by an instance of FIX-ACTIONS$^k_{\text{ADD}}$, respectively. For simplicity, we only consider the case where $tn$ contains only one fact $f$, and we have shown in the hardness proof of FIX-ACTIONS$^k_{\text{ADD}}$ that it is NP-complete even in such a case. To construct an equivalent instance of FIX-ACTIONS$^k_{\text{PREC,ADD}}$, we keep the $k$ unchanged, and construct the action sequence as follows: For each $p_i$ with $1 \leq i \leq n$ and $f \in prec(p_i)$, we construct a sequence of dummy actions $d_{i,1} \cdots d_{i,r}$ where $r = \min\{k, 2|\mathcal{F}||tn|\}$, and for each $1 \leq j \leq r$, $prec(d_{i,j}) = \{f\}$, $del(d_{i,j}) = \emptyset$, and $add(d_{i,j}) = \emptyset$, and place this sequence right before $p_i$ in $tn$. After consulting the argument in the membership proof, one can immediately observe that $r$ is a sufficient upper bound for the maximal number of changes required for both the FIX-ACTIONS$^k_{\text{ADD}}$ instance and the FIX-ACTIONS$^k_{\text{PREC,ADD}}$ instance. By construction, we make FIX-PREC pointless, because if we apply FIX-PREC to some action $p_i$ with $f \in prec(p_i)$, it should also be applied to the sequence $d_{i,1} \cdots d_{i,r}$ accordingly, thus the number of changes applied will exceed $r$. Thereby, the given FIX-ACTIONS$^k_{\text{ADD}}$ instance has a *yes* answer if and only if the instance we constructed has one. □

**Theorem 12.** FIX-ACTIONS$^k_{\text{ADD,DEL}}$ *is NP-complete.*

*Proof. Membership:* Let $tn = p_1 \cdots p_n$ be the given action sequence, and $\mathcal{F} = \bigcup_{i=1}^n prec(p_i) \cup add(p_i) \cup del(p_i)$. $2|\mathcal{F}||tn|$ is a sufficient upper bound for the number of changes making $tn$ executable, which shows membership.

*Hardness:* The argument is almost identical to that in the hardness proof of Thm. 11 except that each dummy action sequence $d_{i,1} \cdots d_{i,r}$ is now placed right after an action $p_i$ in $tn$ with $f \in del(p_i)$, and for each $1 \leq j \leq r$, $prec(d_{i,j}) = \emptyset$, $del(d_{i,j}) = \{f\}$, and $add(d_{i,j}) = \emptyset$. By construction, we make FIX-DEL pointless. Thus, the given FIX-ACTIONS$^k_{\text{ADD}}$ instance has an *yes* answer if and only if the instance we constructed has one. □

**Theorem 13.** FIX-ACTIONS$^k_{\text{PREC,ADD,DEL}}$ *is NP-complete.*

*Proof. Membership:* Let $tn = p_1 \cdots p_n$ be the given action sequence, and $\mathcal{F} = \bigcup_{i=1}^n prec(p_i) \cup add(p_i) \cup del(p_i)$. The number of changes can be bounded by $3|\mathcal{F}||tn|$. Membership follows immediately.

*Hardness:* The argument is similar to that in the proofs of Thm. 11 and 12. We reduce from a FIX-ACTIONS$^k_{\text{ADD}}$ instance where only one fact $f$ is involved. Let $k$ and $tn = p_1 \cdots p_n$ be the integer and the action sequence given by the FIX-ACTIONS$^k_{\text{ADD}}$ instance, respectively. To construct an equivalent instance of FIX-ACTIONS$^k_{\text{PREC,ADD,DEL}}$, we keep $k$ unchanged, and construct the action sequence as follows. For each $p_i$ with $1 \leq i \leq n$ and $f \in prec(p_i)$, we place a dummy action sequence $d_{i,1} \cdots d_{i,r}$ with $r = \min\{k, 3|\mathcal{F}||tn|\}$, and $prec(d_{i,j}) = \{f\}$, $del(d_{i,j}) = \emptyset$, and $add(d_{i,j}) = \emptyset$ for each $1 \leq j \leq r$ right before $p_i$. In the mean time, for each $p_i$ with $1 \leq i \leq n$ and $f \in del(p_i)$, we place a dummy action sequence $d'_{i,1} \cdots d'_{i,r}$ with $prec(d'_{i,j}) = \emptyset$, $del(d'_{i,j}) = \{f\}$, and $add(d'_{i,j}) = \emptyset$ for each $1 \leq j \leq r$ right after $p_i$. By construction, we make both FIX-PREC and FIX-DEL redundant. Thus, the given FIX-ACTIONS$^k_{\text{ADD}}$ instance has a *yes* answer if and only if the FIX-ACTIONS$^k_{\text{PREC,ADD,DEL}}$ instance we constructed has one. □

We can summarize our findings as follows:

**Corollary 3.** *Let* $X \subseteq \{\text{PREC}, \text{ADD}, \text{DEL}\}$ *and* $|X| \geq 1$. FIX-ACTIONS$^k_X$ *is NP-complete if* ADD $\in X$*, otherwise it is in P.*

## 5 Conclusion

Motivated by MIP, modeling assistance, and providing counter-factual explanations for failed plan verification, we investigated the computational complexity of checking whether there exists a sequence of model changes (possibly of bounded length) to turn a given action sequence into a solution. These changes are either performed 1) on decomposition methods, or 2) on the actions' preconditions and effects. For the former, we show that deciding whether such a sequence exists is NP-complete no matter what or how many changes are allowed (unless we are given a sequence of methods where each method refines a unique task, which is in P). For the latter, the problem becomes NP-hard whenever it is allowed to change actions' add lists, otherwise the problem will be in P. A natural exploitation of our results is to implement the decision problems in suitable frameworks, e.g., by relying on *SAT* or *ILPs*, which are both efficient reasoning frameworks for NP-complete problems.

## References


[Ai-Chang *et al.*, 2004] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. C. Hsu, A. Jónsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. G. Chafin, W. C. Dias, and P. F. Maldague. MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*, 19(1):8–12, 2004.

[Alford *et al.*, 2009] R. Alford, U. Kuter, and D. Nau. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *IJCAI*, pages 1629–1634. AAAI, 2009.

[Alford *et al.*, 2015] R. Alford, P. Bercher, and D. Aha. Tight bounds for HTN planning. In *ICAPS*, pages 7–15. AAAI, 2015.

[Allen and Ferguson, 2002] J. Allen and G. Ferguson. Human-machine collaborative planning. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, pages 1–10, 2002.

[Barták *et al.*, 2018] R. Barták, A. Maillard, and R. C. Cardoso. Validation of hierarchical plans via parsing of attribute grammars. In *ICAPS*, pages 11–19. AAAI, 2018.

[Barták *et al.*, 2020] R. Barták, S. Ondrčková, A. Maillard, G. Behnke, and P. Bercher. A novel parsing-based approach for verification of hierarchical plans. In *ICTAI*, pages 118–125. IEEE, 2020.

[Behnke and Speck, 2021] G. Behnke and D. Speck. Symbolic search for optimal total-order HTN planning. In *AAAI*. AAAI, 2021.

[Behnke *et al.*, 2015] G. Behnke, D. Höller, and S. Biundo. On the complexity of HTN plan verification and its implications for plan recognition. In *ICAPS*, pages 25–33. AAAI, 2015.

[Behnke *et al.*, 2016] G. Behnke, D. Höller, P. Bercher, and S. Biundo. Change the plan – how hard can that be? In *ICAPS*, pages 38–46. AAAI, 2016.

[Behnke *et al.*, 2017] G. Behnke, D. Höller, and S. Biundo. This is a solution! (... but is it though?) – verifying solutions of hierarchical planning problems. In *ICAPS*, pages 20–28. AAAI, 2017.

[Behnke *et al.*, 2018] G. Behnke, D. Höller, and S. Biundo. totSAT-totally-ordered hierarchical planning through SAT. In *AAAI*, pages 6110–6118. AAAI, 2018.

[Bercher *et al.*, 2019] P. Bercher, R. Alford, and D. Höller. A survey on hierarchical planning – one abstract idea, many concrete realizations. In *IJCAI*, pages 6267–6275. IJCAI, 2019.

[Bresina *et al.*, 2005] J. L. Bresina, A. K. Jónsson, P. H. Morris, and K. Rajan. Activity planning for the mars exploration rovers. In *ICAPS*, pages 40–49. AAAI, 2005.

[Chakraborti *et al.*, 2017] T. Chakraborti, S. Sreedharan, Y. Zhang, and S. Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*, pages 156–163. IJCAI, 2017.

[Chakraborti *et al.*, 2020] T. Chakraborti, S. Sreedharan, and S. Kambhampati. The emerging landscape of explainable automated planning & decision making. In *IJCAI*, pages 4803–4811. IJCAI, 2020.

[Erol *et al.*, 1996] K. Erol, J. Hendler, and D. S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[Ferguson and Allen, 1998] G. Ferguson and J. F. Allen. TRIPS: an integrated intelligent problem-solving assistant. In *AAAI*, pages 567–572. AAAI, 1998.

[Ferguson *et al.*, 1996] G. Ferguson, J. F. Allen, and B. W. Miller. TRAINS-95: towards a mixed-initiative planning assistant. In *AIPS*, pages 70–77. AAAI, 1996.

[Geier and Bercher, 2011] T. Geier and P. Bercher. On the decidability of HTN planning with task insertion. In *IJCAI*, pages 1955–1961. AAAI, 2011.

[Ginsberg, 1986] M. L. Ginsberg. Counterfactuals. *Artificial Intelligence*, 30(1):35–79, 1986.

[Göbelbecker *et al.*, 2010] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel. Coming up with good excuses: What to do when no plan can be found. In *ICAPS*, pages 81–88. AAAI, 2010.

[Höller *et al.*, 2014] D. Höller, G. Behnke, P. Bercher, and S. Biundo. Language classification of hierarchical planning problems. In *ECAI*, pages 447–452. IOS, 2014.

[Höller *et al.*, 2016] D. Höller, G. Behnke, P. Bercher, and S. Biundo. Assessing the expressivity of planning formalisms through the comparison to formal languages. In *ICAPS*, pages 158–165. AAAI, 2016.

[Karp, 1972] R. M. Karp. Reducibility among combinatorial problems. In *Proc. of a Symposium on the Complexity of Computer Computations*, pages 85–103. Plenum, 1972.

[Korte and Vygen, 2008] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2008.

[Marthi *et al.*, 2007] B. Marthi, S. Russel, and J. Wolfe. Angelic semantics for high-level actions. In *ICAPS*, pages 232–239. AAAI, 2007.

[Masek and Paterson, 1980] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[Myers *et al.*, 2003] K. L. Myers, P. A. Jarvis, W. M. Tyson, and M. J. Wolverton. A mixed-initiative framework for robust plan sketching. In *ICAPS*, pages 256–265. AAAI, 2003.

[Olz *et al.*, 2021] C. Olz, S. Biundo, and P. Bercher. Revealing hidden preconditions and effects of compound HTN planning tasks – a complexity analysis. In *AAAI*. AAAI, 2021.

[Schreiber *et al.*, 2019] D. Schreiber, D. Pellier, H. Fiorino, and T. Balyo. Tree-REX: SAT-based tree exploration for efficient and high-quality HTN planning. In *ICAPS*, pages 382–390. AAAI, 2019.