Change the Plan – How Hard Can That Be?

Gregor Behnke and **Daniel Höller** and **Pascal Bercher** and **Susanne Biundo**
Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany
{gregor.behnke, daniel.hoeller, pascal.bercher, susanne.biundo}@uni-ulm.de

Abstract

Interaction with users is a key capability of planning systems that are applied in real-world settings. Such a system has to be able to react appropriately to requests issued by its users. Most of these systems are based on a generated plan that is continually criticised by him, resulting in a mixed-initiative planning system. We present several practically relevant requests to change a plan in the setting of hierarchical task network planning and investigate their computational complexity. On the one hand, these results provide guidelines when constructing algorithms to execute the respective requests, but also provide translations to other well-known planning queries like plan existence or verification. These can be employed to extend an existing planner such that it can form the foundation of a mixed-initiative planning system simply by adding a translation layer on top.

Introduction

Planning systems deployed to real-world applications ought to interact with their users in order to integrate their wishes and preferences into the plans they generate. There are several possible ways to achieve this integration. Either the user specifies some metric on plans (Fox and Long 2003), e.g., by providing action costs, or he specifies his preferences using a dedicated formalism (Sohrabi, Baier, and McIlraith 2009; Sohrabi and McIlraith 2008). We argue that these approaches might fail in many real-world scenarios, most frequently because the user is not able to specify a-priori all restrictions and preferences he wants to impose on the solution. This can either be caused by the employed planning formalism not being capable of expressing them or the fact that new requirements frequently arise if a solution is presented to him. This gives rise to the paradigm of mixed-initiative planning, which aims at integrating the user into the planning process itself. An established method to do so is to repeatedly present the user with a plan (i.e. a solution to the planning problem), ask his opinion about the plan, and if he is not content to modify it appropriately. Such schemes were, e.g., applied in route-planning (Ferguson et al. 1996), planning of Mars-rover activities (Ai-Chang et al. 2004), and military mission planning (Myers et al. 2003). Arguably, the planning formalism employed by a mixed-initiative planning system should be similar to the one used by its users to ease communication and interaction. In this paper we study mixed-initiative planning systems that employ hierarchical task network (HTN) planning (Erol, Hendler, and Nau 1996). In HTN planning a plan must in addition to being executable also be obtained by repeatedly refining abstract actions into more concrete courses of action. HTN domains are often a natural way to describe the domain in real-world planning applications (and thus particularly in mixed-initiative planning). It is usually easier for modellers to think in terms of hierarchical domains, as humans tend solve problems in a top-down fashion (Byrne 1977; Fox 1997, Sec. 2). HTN domains are also well suited to encode domain-specific search strategies – reducing practical complexity due to search although its theoretical complexity (e.g. in terms of plan existence) becomes harder.

The most critical part of a mixed-initiative planning system is handling the modification request posed by the user. Based on the experiences with MAPGEN – the Mars-rover planning system (Ai-Chang et al. 2004) – Smith (2012; 2013) presented a collection of requests to change plans in a classical setting. More specifically, we study some of the proposed requests and their HTN counterparts by investigating their computational complexity. We will show that, while most of the classical requests can be addressed in polynomial time or **PSPACE**, the same problems are computationally more difficult for HTN planning problems, ranging from **NP** to undecidable. In addition, the proofs throughout this paper provide translations of the user's requests to standard planning queries like plan existence, plan verification, or finding an executable linearisation of a partially ordered plan. Insights into the complexity of user requests is also useful in general, as it enables the designer of a mixed-initiative planning system to select appropriate algorithms for dealing with them. Most notably, one should be aware of the fact that certain request the users can pose are in general undecidable, i.e., there should be a mechanism that handles this case or switches to a decidable problem after a certain time.

We start with a brief introduction for the formalism of HTN planning followed by a discussion on how to interpret the user's requests correctly in the case of HTN planning. Thereafter we provide an investigation of a request that has

been previously studied in the field of mixed-initiative HTN planning: plan sketching (Myers et al. 2003). Lastly, several request based on Smith's (2012; 2013) description are discussed.

Hierarchical planning

We start by giving an introduction to HTN planning (Erol, Hendler, and Nau 1996) using a simplistic formalism proposed by Geier and Bercher (2011). The central concept of HTN planning is the task network, describing a partially ordered set of tasks. Since a task network can contain the same action more than once, we distinguish between a *task* – a unique identifier – and its *task name* – describing the name of the respective action, e.g., *move-a-b*.

Definition 1 (Task Network) *A* task network tn *over a set of task names X is a tuple $(T, \prec, \alpha)$, where*

- *T is a finite, possibly empty, set of tasks*
- *$\prec \subseteq T \times T$ is a strict partial order on T*
- *$\alpha : T \to X$ labels every task with a task name*

TN_X is defined as the set of all task networks over the task names X. As a short-hand notation, we write $T(\mathrm{tn}) = T$, $\prec(\mathrm{tn}) = \prec$ and $\alpha(\mathrm{tn}) = \alpha$ for a task network $\mathrm{tn} = (T, \prec, \alpha)$. Further we define $\mathrm{tn}(x) = (\{1\}, \emptyset, \{(1, x)\})$ to be the task network that contains exactly one task with the task name x. Two task networks $\mathrm{tn} = (T, \prec, \alpha)$ and $\mathrm{tn}' = (T', \prec', \alpha')$ are *isomorphic*, written $\mathrm{tn} \cong \mathrm{tn}'$, if and only if there exists a bijection $\sigma : T \to T'$, such that $\forall t, t' \in T$ it holds that $(t, t') \in \prec$ if and only if $(\sigma(t), \sigma(t')) \in \prec'$ and $\alpha(t) = \alpha'(\sigma(t))$. To ease notation, we define restrictions on relations, functions and task network using the bar notation.

Definition 2 (Restriction) *Let $R \subseteq D \times D$ be a relation, $f : D \to V$ a function and* tn *be a task network. Then the restrictions of R and f to some set X are defined by*

$$R|_X := R \cap (X \times X)$$
$$f|_X := f \cap (X \times V)$$
$$\mathrm{tn}|_X := (T(\mathrm{tn}) \cap X, \prec(\mathrm{tn})|_X, \alpha(\mathrm{tn})|_X)$$

Based upon the definition of a task network, an HTN planning problem is defined as follows.

Definition 3 (Planning Problem) *A* planning problem *is a 6-tuple $\mathcal{P} = (L, C, O, M, c_I, s_I)$, with*

- *L, a finite set of* proposition symbols
- *C, a finite set of* compound task names
- *O, a finite set of* primitive task names *with $C \cap O = \emptyset$*
- *$M \subseteq C \times TN_{C \cup O}$, a finite set of* decomposition methods
- *$c_I \in C$, the* initial task name *and $s_I \in 2^L$, the* initial state

For each primitive task name $o \in O$, its operator *or* action *is given by a tuple that defines its precondition and its effect, the latter in terms of an add-, and a delete list: $(prec(o), add(o), del(o)) \in 2^L \times 2^L \times 2^L$.*

The original formalism of Erol, Hendler, and Nau (1996) allows for an initial task network instead of a single initial task. Since the former version can be compiled into the latter, more simplistic one, we employ the latter. In order to obtain a valid plan in HTN planning, one starts with the initial compound task and repeatedly applies *decomposition methods* to compound tasks until all tasks in the current task network are primitive.

Definition 4 (Decomposition) *A method $m = (c, \mathrm{tn}_m) \in M$ decomposes a task network $\mathrm{tn}_1 = (T_1, \prec_1, \alpha_1)$ into a task network tn_2 by replacing task t, written $\mathrm{tn}_1 \xrightarrow[t,m]{} \mathrm{tn}_2$, if and only if $t \in T_1$, $\alpha_1(t) = c$, and $\exists \mathrm{tn}' = (T', \prec', \alpha')$ with $\mathrm{tn}' \cong \mathrm{tn}_m$ and $T' \cap T_1 = \emptyset$, where*

$$\mathrm{tn}_2 := (T'', \prec_1 \cup \prec' \cup \prec_X, \alpha_1 \cup \alpha')|_{T''} \text{ with}$$
$$T'' := (T_1 \setminus \{t\}) \cup T'$$
$$\prec_X := \{(t_1, t_2) \in T_1 \times T' \mid (t_1, t) \in \prec_1\} \cup \{(t_1, t_2) \in T' \times T_1 \mid (t, t_2) \in \prec_1\}$$

We write $\mathrm{tn}_1 \to_D^* \mathrm{tn}_2$, if tn_1 can be decomposed into tn_2 using an arbitrary number of decompositions.

The original formulation of HTN planning by Erol, Hendler, and Nau (1996) and the simplistic reformulation by Geier and Bercher (2011) defined a task network to be executable, if there *exists* a linearisation of its tasks that is executable. The related formalisms hybrid planning (Biundo and Schattenberg 2001) and POCL planning (Penberthy and Weld 1992), on the other hand require all linearisations to be executable. We will restrict our investigation to the former formalism.

Definition 5 (Executable Task Network) *A task network $(T, \prec, \alpha)$ is* executable in a state $s \in 2^L$*, if and only if all its tasks are primitive and there exists a linearisation of its tasks $t_1, \ldots, t_n$ that is compatible with $\prec$ and a sequence of states $s_0, \ldots s_n$ such that $s_0 = s$ and $prec(\alpha(t_i)) \subseteq s_{i-1}$ and $s_i = (s_{i-1} \setminus del(\alpha(t_i))) \cup add(\alpha(t_i))$ for all $1 \leq i \leq n$.*

Using the previous definition we can describe the set of solutions to a planning problem $\mathcal{P}$. We deviate from the definition given by Geier and Bercher (2011), in that we allow a solution to have more ordering constraints than those required by the decomposition it was generated by. This is done to prevent problematic interactions with users, who are presented with a solution and ask to add an ordering constraint, but the planner refuses to do so, while the plan is still executable and can clearly be generated by decomposition. For example, a solution might contain the actions a and b unordered and both linearisations are executable. If we use the original solution criterion, the task network where a is ordered strictly before b is not a solution, since the additional ordering constraint $a < b$ cannot be derived from the decomposition hierarchy, but is only compatible with it. Such a case seems to be extremely confusing for a user and should be avoided.

Definition 6 (Solution) *A task network tn_S is a solution to a planning problem $\mathcal{P}$, if and only if*

(1) *tn_S is executable in s_I,*

(2) *$\mathrm{tn}(c_I) \to_D^* \mathrm{tn}_D$,*

(3) *$\mathrm{tn}_S = (T(\mathrm{tn}_D), \prec', \alpha(\mathrm{tn}_D))$ where $\prec(\mathrm{tn}_D) \subseteq \prec'$.*

$\mathrm{Sol}(\mathcal{P})$ denotes the sets of all solutions of $\mathcal{P}$.

HTN planning semantics – how should we interpret the domain when changing a plan?

In the previous section we have outlined the criteria a task network must fulfil in order to be considered a solution to an HTN planning problem: it has to be executable and it must reside in the refinement space of the HTN problem, i.e., it must be possible to obtain it through a sequence of decompositions and ordering insertions form the initial task network. If such a plan is presented to the user and he utters a request to change the plan, we have to consider what we deem an appropriate solution. Obviously, the plan must comply with the request itself, but the question arises which of the solution criteria for HTN planning the changed plan must also fulfil. While undoubtedly executability must be preserved, the question whether it must also lie with the refinement space has to be discussed.

If we only require the changed plan to be executable, we essentially view the HTN domain as a mere guide or heuristic for the search process in order to obtain a solution. Further this allows the planner, when changing a plan upon the user's request, to ignore the knowledge and semantics contained within the HTN structure of the domain completely. This poses a problem, since HTN planning problems ordinarily do not specify a goal state to be reached, but aim at performing a certain set of (compound) actions (Erol, Hendler, and Nau 1996). This makes the task hierarchy an integral part of the problem description and hence restricts the set of allowed solutions. If it would be ignored, any executable sequence of actions would be considered a solution, which seems not to be appropriate.

One possible interpretation of an HTN domain – apart from providing a structural guide for the search – is that it provides a more expressive planning formalism. Ignoring the decomposition hierarchy also ignores criteria for executability (or otherwise the plan's validity) that are not encoded as preconditions and effects but rather as decomposition methods. Earlier work has shown that HTN planning is strictly more expressive than classical planning under two separate views, which shows that such criteria exist: On the one hand, deciding whether a given planning problem has a solution is undecidable (Erol, Hendler, and Nau 1996; Geier and Bercher 2011), which entails that there is no way to translate an arbitrary HTN planning problem into a classical planning problem[1]. On the other hand, there are results about the structural restrictions the planning problem can possibly enforce upon its solutions. We (2014; 2016) showed that the structures of HTN solutions are strictly more expressive than solutions for classical problems. This enables domain modellers to express restrictions to the plans and thus the "state changes" in the scenario using the hierarchy, which are not describable by preconditions and effects on a finite set of literals.

To conclude, we deem it necessary and appropriate to require that a plan lies within the problem's refinement space and thus still satisfies the respective hierarchical solution criterion, in order to ensure that the changed plan complies with the restrictions of the planner's application domain. This means that whenever a plan has been changed, we (at least) have to verify that it is a plan in the HTN sense.

[1]However there are translations for several more restrictive subclasses of HTN domains which are commonly found in application domains (Alford et al. 2016; Alford, Kuter, and Nau 2009).

Planning with given actions

We start our investigation of user requests posed to mixed-initiative planning systems by looking into a specific type of request that has been discussed in prior work. Myers (1997) proposed a mechanism called "plan sketching" handling "must contain" constraints for HTN planning. Her approach lets the user provide a set of actions and ensures that the generated solution contains these actions. In addition to requirements on primitive tasks, i.e., those actually contained in the solution, her mechanism also allows for requirements on compound tasks, i.e., forcing that the solution has been obtained such that these tasks were intermediate steps. A precise definition of a *plan sketch* and *plan sketch compliance* is given below. The intuition is that if a compound task c is contained in the sketch, a plan complies with it if it is possible to decompose the initial task network such that c occurs at some point during decomposition in the current task network.

Definition 7 *Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem and $\mathcal{S} \subseteq C \cup O$ a* sketch*. A task network $\mathrm{tn}^* \in \mathrm{Sol}(\mathcal{P})$ complies with $\mathcal{S}$ if there is a sequence of task networks, such that*

$$\mathrm{tn}(c_I) = \mathrm{tn}_0 \to_D^* \mathrm{tn}_1 \to_D^* \ldots \to_D^* \mathrm{tn}_n = \mathrm{tn}^*$$

and $S \subseteq \bigcup_{i=0}^{n} \bigcup_{t \in T(\mathrm{tn}_i)} \alpha(\mathrm{tn}_i)(t)$

We call the problem, given an HTN planning problem $\mathcal{P}$ and a sketch $\mathcal{S}$, to determine whether $\mathcal{P}$ has a solution that complies with S, SKETCH-COMPLIANCE. Based upon earlier results by Erol, Hendler, and Nau (1996), we can prove that this problem is strictly semi-decidable.

Theorem 1 *The problem* SKETCH-COMPLIANCE *is strictly semi-decidable.*

Proof: Undecidability: If the sketch $\mathcal{S}$ is chosen as the empty set, the problem is equivalent to the plan existence problem, which is known to be undecidable (Erol, Hendler, and Nau 1996; Geier and Bercher 2011).

Enumerability: We can transform a SKETCH-COMPLIANCE instance into an ordinary plan existence problem and use the property that the set of solutions to a given planning problem is enumerable (Erol, Hendler, and Nau 1996). Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ be a planning problem and $\mathcal{S}$ a sketch. First we add for every $s \in S$ a new proposition $\overline{s}$ to L. For every $s \in \mathcal{S} \cap O$ we add the add-effect $\overline{s}$ to s. For every $s \in \mathcal{S} \cap C$ we add a new primitive task name n_s without preconditions and the sole add-effect $\overline{s}$. It is added to the task network of every method (s, tn) which can decompose s without any further ordering constraints.

Lastly two new tasks – the primitive task g and the compound task c_I^* – are added. An additional decomposition

method $(c_I^*, \mathrm{tn}_{c_I^*})$ is added for the latter with

$$\mathrm{tn}_{c_I^*} = (\{1,2\}, \{(1,2)\}, \{(1,c_I), \{2,g\}\})$$

where g has $\{\overline{s} \mid s \in S\}$ as its precondition and no effects. In the constructed planning problem, c_I^* is used as the initial task instead of c_I.

Any solution to the new planning problem contains g as its last task and g is executable is this task network. Thus all $\overline{s}$ were made true, implying that for all $s \in \mathcal{S} \cap C$ an instance of n_s was inserted into the plan and for all $s \in \mathcal{S} \cap O$ that the task s is contained in the plan. Consequently the solution complies with the sketch. Also a solution to a SKETCH-COMPLIANCE problem can trivially be transformed into a solution to the just constructed HTN planning problem. □

The proof provides a scheme to compile a SKETCH-COMPLIANCE instance into a standard HTN planning problem. By applying the scheme ordinary HTN planners – like SHOP or UMCP – could be used to solve the problem instead of the specialised algorithm developed by Myers (1997). This would lift the restriction of her algorithm, which is only applicable to so-called acyclic domains. A planning problem is acyclic if for all compound tasks $c \in C$ there is no task network tn^* such that $\mathrm{tn}(c) \rightarrow_D^* \mathrm{tn}^*$ and tn^* contains a task labelled with c, i.e., no compound task can be achieved via decomposition from itself. Formally an HTN planning problem is defined as follows

Definition 8 *An* HTN *planning problem* $\mathcal{P} = (L, C, O, M, c_I, s_I)$ *is acyclic if a total order* $<$ *on* C *exists such that for all methods* $(c, \mathrm{tn}) \in M$ *all compound tasks in* tn *are smaller than* c *according to* $<$.

In addition to the fact that the algorithm only treats acyclic problems, she does not provide any bound on the complexity of the algorithm (only for parts thereof). Using the proof of the theorem – which does not introduce cycles into the domain – and the result by Alford, Bercher, and Aha (2015), stating that plan existence for acyclic planning problems is **NEXPTIME-complete**, we can conclude the following corollary, which enables an estimation of the actual complexity of her algorithm.

Corollary 1 *The problem* SKETCH-COMPLIANCE *for acyclic planning problems is* **NEXPTIME-*complete***.

Intuitively, a user might also pose the request to avoid certain actions when generating a plan. The respective decision problem would ask, whether given a set A of task names there is a solution that does not contain any element of A. This problem can trivially be solved by simply removing all tasks in A from the domain. The proof of the following proposition is straightforward and is thus omitted.

Proposition 1 *The problem* ACTION-AVOIDING *is strictly semi-decidable.*

Changing plans

Plan sketching enables a user to pose a specific kind of restrictions on the actions prior to the planner generating a plan, which is a useful instrument in mixed-initiative planning. It does not cover cases where a plan has already been generated and presented to the users, which should be considered when generating a solution. In practice this is not sufficient, since – as argued before – the planning system has also to integrate requests to change an already generated plan. Such requests include, e.g., the wish to add certain actions or to reorder them in a specific way. In this section we discuss several practically motivated requests, based on a talk given by Smith (2012; 2013). We provide formal definitions of these requests for HTN planning problems and study their computational complexity. Handling of most of these requests is already available in mixed-initiative planning systems for classical planning, like MAPGEN (Ai-Chang et al. 2004). In the case of HTN planning there are – to our knowledge – no systems available that can handle the requests presented.

While we focus on the computational complexity of those requests, the respective membership proofs often provide schemes to compile them into standard enquiries usually posed to an HTN planning system, like plan existence. Using these schemes it would be straightforward to implement the requests in any state-of-the-art planning system. This enables a mixed-initiative planning system's designer to employ a standard planning algorithm without the need to change or adapt it to the mixed-initiative system it is used in. Similarly, it is rather easy to extend the capabilities of planners, in terms of the enquiries they can handle, significantly by adding a translation layer on-top of the actual planning routine.

Most of the results in this section are based on a result of Behnke, Höller, and Biundo (2015), who showed that determining whether a given task network is a solution to a given planning problem is **NP-complete**. This problem is commonly also called plan verification.

Definition 9 *The problem* VERIFY-TN *is, given a planning problem* $\mathcal{P}$ *and a task network* tn*, to decide whether* $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$.

Proposition 2 VERIFY-TN *is* ***NP-complete***.

Moreover they showed that the problem is even **NP-complete** if the planning problem is *totally unordered*, i.e., all task networks in the planning domain do not contain any ordering constraints ($\forall (c, \mathrm{tn}) \in M : \alpha(\mathrm{tn}) = \emptyset$) and all primitive actions are precondition-free. This means that only the number of instances of each primitive action in a task network matters when determining whether it is a solution to the planning problem or not.

Changing Order

We start with an enquiry which we expect a user to make quite often: the request to change the order of the actions in a given plan, uttered e.g. as "Can I rearrange the tasks such that ...?". For example it might be requested that a mars rover first has to take a picture of its surrounding before it can take a surface sample, but the planner has arranged the actions the other way round to conserve energy. The planner's task is to determine whether it is possible to comply with the change while remaining enough energy to operate safely.

Obviously, there are several varieties of the problem depending on which kinds of ordering constraints he is allowed to enforce and on whether the planner can make additional changes to the plan and if so to which extent. We restrict our investigation to requests containing only a single ordering constraint, as extending the presented results to cases involving multiple constraints or disjunctive requests (like "Establish that either action a or action b is before c.") is straightforward. We assume in our formulation that the user asks for the ordering $t_1 < t_2$ to be established in the current plan.

We start with the most restrictive version, where we forbid the planner to alter the plan in any way, except the change explicitly requested by the user.

Definition 10 *We define the problem* ORDER *as:*
Given a planning problem $\mathcal{P}$, a task network $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$, and two tasks $t_1, t_2 \in T(\mathcal{P})$, is the task network $(T(\mathrm{tn}), \prec(\mathrm{tn}) \cup \{(t_1, t_2)\}, \alpha(\mathrm{tn}))$ also a solution to $\mathcal{P}$?

Surprisingly deciding this problem is already **NP-complete** as stated by the following theorem. However, this is only based on a peculiarity of the HTN formalism, which requires a solution to have only a single executable linearisation and not that all linearisations to be executable. Inserting an additional ordering constraint might exclude this linearisation and requires to determine whether another executable linearisation exists. If the criterion for a solution would require that all linearisations are executable (which can be determined in polynomial time (Nebel and Bäckström 1994)) this request would therefore be in **P**. The respective request in classical planning is to reorder a plan in a given way. This is trivially polynomial, since the change can be performed and executability be tested.

Theorem 2 ORDER *is **NP-complete**.*

Proof: Membership: Given the task network tn, we first have to check whether the ordering $t_2 < t_1$ is already contained in tn. If so, the answer to the problem is no. Else, we can insert the ordering and obtain tn'. Since tn is a solution, tn' is also reachable from c_I via decomposition and insertion of ordering constraints. What remains is to check whether tn' has an executable linearisation, which can be done by guessing one and checking it.

Hardness: We can reduce from the problem of determining whether a task network has an executable linearisation which in turn is known to be **NP-complete** (Erol, Hendler, and Nau 1996; Nebel and Bäckström 1994, Thm. 14 & 15). Given a task network tn we can construct a planning domain as follows. First we add for every task name n occurring in tn a new primitive task a_n, which has no preconditions and the preconditions of n as its effects. Further two new primitive tasks e and b are added to the problem. Next we construct the task network tn^* which will be used as the input to the ORDER problem. It contains the tasks of tn with the same order, one instance of e and b and and the appropriate amount of a_n tasks to ensure that tn^* is executable. Formally we define

$$\mathrm{tn}^* = (T(\mathrm{tn}) \cup T^*, \prec(\mathrm{tn}) \cup \prec^*, \alpha(\mathrm{tn}) \cup \alpha^*) \quad \text{where}$$
$$T^* = \{1, 2\} \cup \{\bar{t} \mid t \in T(\mathrm{tn})\}$$
$$\prec^* = \{(t, 1), (2, \bar{t}) \mid t \in T(\mathrm{tn})\}$$
$$\alpha^* = \{(1, e), (2, b)\} \cup \{(\bar{t}, n_{\alpha(\mathrm{tn})(t)}) \mid t \in T(\mathrm{tn})\}$$

The only decomposition method in the planning problem is (c_I, tn^*). The question to be posed is whether it is possible to insert the ordering $1 < 2$ into the task network tn.

The initial task network tn^* is clearly executable as for any task $t \in T(\mathrm{tn})$ the respective task $\bar{t}$ could just be ordered right before t ensuring that it is executable. If the ordering $1 < 2$ is inserted into the plan, these tasks cannot be used any more to ensure the executability of any task $t \in T(\mathrm{tn})$. Thus this task network is only a solution to the planning problem if the original task network tn has an executable linearisation. □

In most practical cases however requiring that none of the other ordering constraints can be altered seems to be overly restrictive. Alternatively the planner can be allowed to add new ordering constraints or to remove some of those present in the current plan. In general, as few orderings as possible should be changed to keep the plan as similar to the original plan as possible. This seems necessary to enable the user to understand the performed changes and the structure of the new plan easily. Only if no plan fulfilling the given restrictions exists, the planner might weaken them with a proper warning to the user.

The problem is to determine the minimal number of ordering constraints that have to be changed in order to allow for the ordering $t_1 < t_2$ to be inserted into the task network. Apparently the more general problem to determine whether it is possible at all to rearrange the tasks of a task network compliant to $t_1 < t_2$ such that the task network is a solution is **NP-complete**. From this we can easily deduce **NP-completeness** of the minimisation problem.

Definition 11 *We define the problem* REORDER *as:*
Given a planning problem $\mathcal{P}$, a task network $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$, and two tasks $t_1, t_2 \in T(\mathcal{P})$, is there a partial ordering $\prec$ such that $(T(\mathrm{tn}), \prec, \alpha(\mathrm{tn}))$ is a solution to $\mathcal{P}$ and the ordering $t_1 < t_2$ is contained in $\prec$?

Theorem 3 REORDER *is **NP-complete**.*

Proof: Membership: A partial order $\prec$ containing $t_1 < t_2$ can be guessed and the resulting task network can be checked using the **NP** algorithm for VERIFY-TN. Note that this does not result in a $\mathbf{NP}^{\mathbf{NP}}$[2] algorithm, since the application of the VERIFY-TN algorithm is the last step and its result will be returned without alteration.

Hardness: We reduce from the VERIFY-TN problem for totally unordered task networks without preconditions. Let tn be a task network to be verified and $\mathcal{P}$ be the respective problem. If tn contains any ordering constraints, it cannot be a solution of $\mathcal{P}$. We alter $\mathcal{P}$ as follows. First a new primitive

[2] $\mathbf{NP}^{\mathbf{NP}}$ denotes the class of problems solvable via an **NP** algorithm that can query another **NP** algorithm at every step.

task a, without any preconditions or effects is added to $\mathcal{P}$. Second we replace the initial abstract task c_I with the new c'_I for which we define the following two methods

$$m_1 = (c'_I, (\{1,2\}, \{(1,2)\}, \{(1,c_I),(2,a)\}))$$
$$m_2 = (c'_I, (T(\mathrm{tn}) \cup \{1\}, \{(1,t) \mid t \in T(\mathrm{tn})\}, \alpha(\mathrm{tn}) \cup \{(1,a)\}))$$

The question put to the REORDER solver is whether it is possible to reorder the task network of m_2 such that $t < 1$ for an arbitrary $t \in T(\mathrm{tn})$. The task network in question is clearly a solution, and if the order $t < 1$ is to be fulfilled, the method m_1 has to be applied in order to decompose c'_I. Thus if it is possible to reorder the tasks, the task network tn was a solution to $\mathcal{P}$. □

In the two requests investigated so far, we were restricted not to alter the actions contained in the task network but were only allowed to rearrange them. In certain situations this might be a too strict constraint, e.g., if a new action has to be inserted to accommodate for the new order. A simple solution would be to allow for arbitrary changes to the task network and requiring that the result contains a given ordering constraint. Since the original task network the user posts its request for only provides the tasks between an ordering constraint shall be established, the actual task network can be disregarded leading to the following problem definition.

A classical version of this request asks the same question, whether there is a plan that contains two actions in a specific order, which can be encoded into the problem with appropriate preconditions and effects. **PSPACE-completeness** can be obtained via a reduction from plan existence.

Definition 12 *We define the problem* ORDER-SOLUTION *as:*
Given a planning problem $\mathcal{P}$ and two task names $n_1, n_2 \in O$, is there a task network $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$ such that there are $t_1, t_2 \in T(\mathrm{tn})$, $\alpha(\mathrm{tn})(t_1) = n_1$, $\alpha(\mathrm{tn})(t_2) = n_2$, and $t_1 < t_2 \in \prec(\mathrm{tn})$.

Unsurprisingly – after considering Thm. 1, this problem is strictly semi-decidable.

Theorem 4 ORDER-SOLUTION *is strictly semi-decidable.*

Proof: Undecidability: We can reduce from the plan existence problem for general HTNs, which is known to be undecidable (Erol, Hendler, and Nau 1996; Geier and Bercher 2011). Let $\mathcal{P}$ be a planning problem. We add three new task names to $\mathcal{P}$: the primitives n_1 and n_2 and the abstract c^*_I, which replaces c_I as the initial abstract task. The sole decomposition method for c^*_I decomposes it into the following task network

$$\mathrm{tn} = (\{1,2,3\}, \{(1,2)\}, \{(1,n_1),(2,n_2),(3,c_I)\})$$

Any solution to the original problem $\mathcal{P}$ translates to a solution of the new problem containing $n_1 < n_2$ and vice versa. Thus enquiring for a solution to the new problem containing $n_1 < n_2$ solves the plan existence problem.

Enumerability: The solutions to $\mathcal{P}$ are enumerable and each solution can be tested whether it contains $n_1 < n_2$. □

In most practical cases the changes that can reasonably be applied to a solution in order to fulfil a given request is small. Without such a restriction the planner would practically be allowed to create a completely new task network. Such a behaviour should be prohibited to ensure a stable presentation of plans to the user.

A simple model to restrict the changes that can be applied to a plan is to bound their number of by some small constant. This leads to the following version of the ORDER problem and the result that it is **NEXPTIME-complete**.

Definition 13 *We define the problem* FORCE-ORDER *as:*
Given a planning problem $\mathcal{P}$, a task network $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$, two tasks $t_1, t_2 \in T(\mathcal{P})$, and an integer k. Can tn be transformed into a solution $\mathrm{tn}' \in \mathrm{Sol}(\mathcal{P})$ with at most k of the following operations

- *adding a new primitive task*
- *removing a task from the task network*
- *adding an ordering constraint*
- *removing an ordering constraint.*

such that neither t_1 nor t_2 are removed and tn' contains the ordering constraint $t_1 < t_2$.

Theorem 5 FORCE-ORDER *is* **NEXPTIME-*complete***.

Proof: Membership: Given a task network tn we can apply up to k of the described operations randomly in exponential time[3]. We can test for the resulting task network whether it is a solution or not and whether it satisfies $t_1 < t_2$ in nondeterministic polynomial time, which leads to a **NEXPTIME** decision procedure.
Hardness: We reduce from the plan existence problem for acyclic HTN planning problems (see Definition 8), which is known to be **NEXPTIME-complete** (Alford, Bercher, and Aha 2015). Let $\mathcal{P}$ be an acyclic HTN planning problem.

First, two new primitive tasks names n_1 and n_2 – without preconditions and effects – and a new compound task c^*_I are added to the domain. The latter will be the initial compound task of the domain. Also the following two decomposition methods (c^*_I, tn_1) and (c^*_I, tn_2) are added, where

$$\mathrm{tn}_1 = (\{1,2\}, \{(1,2)\}, \{(1,n_1),(2,n_2)\})$$
$$\mathrm{tn}_2 = (\{1,2,3\}, \{(2,1)\}, \{(1,n_1),(2,n_2),(3,c_I)\})$$

The question posed is whether it is possible to achieve the order $2 < 1$ in the task network tn_1. This is obviously possible if and only if the original planning problem has a solution. What remains is to determine k appropriately. Since the original planning problem is acyclic any solution contains no more than $p_{max} = \delta^{|O|}$ primitive tasks, where δ is the largest size of a task network in any method. Also any solution contains no more than $\frac{1}{2}p_{max}(p_{max}+1)$ ordering constraints. Since $p_{max} = \delta^{|O|} \leq |\mathcal{P}|^{|\mathcal{P}|} \leq 2^{|\mathcal{P}|^2}$ we can choose k to be an appropriate exponential, e.g.,

[3] Since k is encoded logarithmically, we cannot do this in polynomial time.

$k = 2 \cdot 2^{|\mathcal{P}|^4}$. As k is encoded logarithmically, the encoding can be performed in polynomial time. □

The interested reader might notice that the problem is mainly **NEXPTIME-complete** in the number k – the number of allowed changes. Since one can argue, as done above, that k should have a small value (e.g. ≤ 10), this might not be a problem in practice. This could be justified by further fixed parameter tractability investigations.

Changing actions

Another type of user requests relates directly to the actions of a plan. In this paper we distinguish three varieties of such enquiries, they either ask to add, to remove, or to replace a certain action given a current plan. In this section we restrict the discussion of these problems to the variant similar to the original ORDER problem. Since the proofs for the analogue problems to REORDER, ORDER-SOLUTION, and FORCE-ORDER proceed similarly, we omit them for the sake of brevity.

The seemingly most trivial operation to be performed is to replace some action in a solution with another action and require that the resulting task network is still a solution to the planning problem. A scientist might, e.g., request to replace taking a photo with the ordinary camera with an action using the infra-red camera. In a classical setting this is clearly solvable in polynomial time. One has to replace the action and check whether the resulting plan is still executable. In the case of HTN planning this problem becomes **NP-complete** since we have to verify that the new solution can still be generated by the decomposition hierarchy.

Definition 14 *We define the problem* EXACT-REPLACE *as: Given a planning problem $\mathcal{P}$, a task network $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$, a task $t \in T(\mathrm{tn})$ and a primitive task name $p \in O$, is the task network tn' where t is labelled with p also a solution to $\mathcal{P}$?*

Theorem 6 REPLACE *is **NP-complete**.*

Proof: Membership: By replacing the task name associated with t with p we obtain a new task network tn^*. Using the **NP** algorithm for the VERIFY-TN problem, we can decide whether tn^* is also a solution to the planning problem.

Hardness: We can reduce totally unordered precondition-free VERIFY-TN to the REPLACE problem. Let tn^* be a task network to be verified and $\mathcal{P}$ the respective planning problem. If tn^* is empty, we determine whether c_I can be decomposed into the empty task network in polynomial time using the procedure proposed by Behnke, Höller, and Biundo (2015).

Else, let a be a new primitive task name (without any preconditions or effects) that does not occur in $\mathcal{P}$, i.e., $a \notin O \cap C$. We construct a new task network tn' which is identical to tn^* with the only difference that we replace the task name of an arbitrary task t by a. Let the task name of t in tn^* be b. Finally an additional method (c_I, tn') is added to the planning problem.

The VERIFY-TN problem can now be decided by asking whether b can be replaced by a in tn'. □

Similarly, the users might ask to add an action to a solution, e.g., if performing a chemical analysis of a sample can be added to a rover's plan. For classical planning this task is still polynomial, as every possible position for the new action can be examined and checked. In the HTN setting, however, the question is whether the additional task can be obtained via decomposition in addition to the original task network.

Definition 15 *We define the problem* ADD *as: Given a planning problem $\mathcal{P}$, a task network $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$, and a primitive task name $p \in O$, is there a task network tn' which is tn with an instance of p added such that $\mathrm{tn}' \in \mathrm{Sol}(\mathcal{P})$?*

Theorem 7 ADD *is **NP-complete**.*

Proof: Membership: By adding the task name p to tn we obtain a new task network tn^*. Using the **NP** algorithm for the VERIFY-TN problem, we can decide whether tn^* is also a solution to the planning problem.

Hardness: We reduce VERIFY-TN to the ADD problem. Let tn^* be a task network to be verified and $\mathcal{P}$ the respective planning problem. Let a be a new primitive and c'_I a new compound task name. We add them and the two decomposition methods $m_1 = (c'_I, \mathrm{tn}^*)$ and $m_2 = (c'_I, (\{1,2\}, \emptyset, \{(1, c_I), (2, a)\}))$ to the planning problem.

The VERIFY-TN problem can now be decided by asking whether a can be added to tn^*. □

Lastly, users might also enquire whether it is possible to remove an action from a given plan. This might, e.g., happen if a scientist objects to moving the rover as suggested by the planner, in order to ensure that the rock sample is not polluted by dust created by moving the wheels. The same reasoning as for the ADD request holds, making it polynomial for classical planning and **NP-complete** for HTN planning.

Definition 16 *We define the problem* REMOVE *as: Given a planning problem $\mathcal{P}$, a task network $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$, a task $t \in T(\mathrm{tn})$, is the task network $\mathrm{tn}|_{T(\mathrm{tn}) \setminus \{t\}}$ also a solution to $\mathcal{P}$?*

Theorem 8 REMOVE *is **NP-complete**.*

Proof: Membership: By removing the task t we obtain a new task network tn^*. Using the **NP** algorithm for the VERIFY-TN problem, we can decide whether tn^* is also a solution to the planning problem.

Hardness: We can reduce VERIFY-TN to the REMOVE problem. Let tn^* be a task network to be verified and $\mathcal{P}$ the respective planning problem.

Let a be a new primitive task name that does not occur in $\mathcal{P}$, i.e., $a \notin O \cup C$ and t be a task that does not occur in tn. We construct a new task network $\mathrm{tn}' = (T(\mathrm{tn}) \cup \{t\}, \prec(\mathrm{tn}), \alpha(\mathrm{tn}) \cup \{(t, a)\})$ which is tn but with a new task t with the task name a added. Finally the method (c_I, tn') is added to the planning problem.

The VERIFY-TN problem can now be decided by asking whether t can be obtained from tn'. □

Changing Causality

We allege that in real-world settings users might also want to change the internal causality of a plan. An example for such a request is to change a given plan such that it avoids a specific way to produce an effect.

Suppose we have a rover with two chemical analysis sets for rocks (X and Y) on board. Both can determine the property A of a rock. In addition to determining the property A, the set X can simultaneously determine property B, too, and set Y can also determine the property C. The goal is to know all three properties A,B, and C of a rock. In the initial plan – automatically generated by a planner – both X and Y have to be used and (by chance) X's result for A is used. A scientist now requests not to use X's result as he fears that A's sensor is contaminated from the last rock sample, while Y's sensor isn't. Based upon a formal description of this problem we show that it is in fact **NP-complete**.

Definition 17 *We define the problem* AVOID-EFFECT *as: Given a planning problem $\mathcal{P}$, a task network $\mathrm{tn} \in \mathrm{Sol}(\mathcal{P})$, a task name $n \in P$, and an effect (either add or delete) e of n, is tn also a solution if e would not be an effect of n?*

Theorem 9 STRICT-AVOID-EFFECT *is **NP-complete**.*

Proof: Membership: Given the task network tn, a task name n and an effect e, we can remove the effect e from all instances of n in tn, guess a linearisation of tn and check whether it is executable. Since neither tasks nor orderings are added or removed tn can always be obtained from c_I.

Hardness: The hardness proof is structurally similar to the proof of Thm. 2. We reduce from the problem of determining whether a task network has an executable linearisation (Erol, Hendler, and Nau 1996; Nebel and Bäckström 1994, Thm. 14 & 15). Let tn be a task network for which we have to determine whether it has an executable linearisation.

We construct a new task network tn^* which initially is equal to tn and modify it as follows. First, we add two new actions with names a and b (which do not occur in tn) to tn^* such that a is ordered before all tasks in tn and b after all these tasks. Both tasks have an add effect with the new predicate e, which also does not occur in tn otherwise. Second, a new action n_t is added to tn^* for every task t in tn, which has the preconditions of t as its effects and e as its precondition.

The resulting task network tn^* has an executable linearisation, namely any one where every n_t is ordered just before its task t. We use the STRICT-AVOID-EFFECT problem to determine whether tn^* has an executable linearisation, if the effect e of a should not be used, any task n_t must be ordered strictly after b to be applicable. Thus the task network is executable if and only if the original task network tn has an executable linearisation. □

Conclusion

Handling user requests is a central capability for any mixed-initiative planning system, if it is to be successfully deployed. In this paper, we have presented a comprehensive list of such requests, mainly based on practical experience described by Smith (2012; 2013) and a system of Myers (1997). We started by determining the computational complexity of HTN plan sketching (Myers 1997). As most of the requests proposed by Smith are formulated for classical planning, we have re-formulated them for HTN planning. To do so, we have discussed the role of an HTN planning problem's structures and restrictions when fulfilling the user's requests to change a plan. Taking the request to establish a specific ordering constraint as an example, we have discussed several appropriate variants of the problem, varying in the type and amount of changes to the plan allowed. We proved tight complexity bounds – ranging from **NP-complete** to strict semi-decidability – for all those requests, which can easily be extended to those cases not explicitly discussed in the paper. Furthermore, the membership proofs of those theorems provide a comprehensive list of reductions to other planning enquiries, like plan existence. Using those compilations one can easily add the respective capabilities on-top of an existing planning to obtain the scaffold of a mixed-initiative planning system. As far as we know we can also claim that we have provided a first theoretical foundation of mixed-initiative planning in terms of an investigation of computational complexity.

Acknowledgments

This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; et al. 2004. MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. *Intelligent Systems, IEEE* 19(1):8–12.

Alford, R.; Bercher, P.; and Aha, D. 2015. Tight bounds for HTN planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 7–15. AAAI Press.

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; and Aha, D. 2016. Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press.

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, 1629–1634. AAAI 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 Press.

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In *Proceedings of the 6th European Conference on Planning (ECP)*, 157–168.

Byrne, R. 1977. Planning meals: Problem solving on a real data-base. *Cognition* 5:287–332.

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

Ferguson, G.; Allen, J. F.; Miller, B. W.; et al. 1996. TRAINS-95: Towards a mixed-initiative planning assistant. In *Proceedings Third Conference AI Planning Systems (AIPS)*, 70–77.

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal Artificial Intelligence Research (JAIR)* 20:61–124.

Fox, M. 1997. *Proceedings of the 4th European Conference on Planning (ECP)*. Springer Berlin Heidelberg. chapter Natural hierarchical planning using operator decomposition, 195–207.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 1955–1961.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 447–452.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the expressivity of planning formalisms through the comparison to formal languages. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press.

Myers, K. L.; Jarvis, P.; Tyson, M.; and Wolverton, M. 2003. A mixed-initiative framework for robust plan sketching. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, 256–266.

Myers, K. L. 1997. Abductive completion of plan sketches. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, 687–693. AAAI Press.

Nebel, B., and Bäckström, C. 1994. On the computational complexity of temporal projection, planning, and plan validation. *Artificial Intelligence* 66(1):125–160.

Penberthy, S. J., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR)*. Morgan Kaufmann. 103–114.

Smith, D. E. 2012. Planning as an iterative process. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2180–2185. AAAI Press.

Smith, D. E. 2013. The challenges of interactive science planning. Summer School of the 23rd International Conference on Automated Planning and Scheduling (ICAPS).

Sohrabi, S., and McIlraith, S. A. 2008. On planning with preferences in HTN. In *Proceedings of the 12th International Workshop on Non-Monotonic Reasoning (NMR)*, 241–248.

Sohrabi, S.; Baier, J.; and McIlraith, S. A. 2009. HTN planning with preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 1790–1797.