Tight Bounds for Hybrid Planning

Pascal Bercher[1] , **Songtuan Lin**[1] , **Ron Alford**[2]
[1]The Australian National University
[2]The MITRE Corporation
pascal.bercher@anu.edu.au, songtuan.lin@anu.edu.au, ralford@mitre.org

Abstract

Several hierarchical planning systems feature a rich level of language features making them capable of expressing real-world problems. One such feature that's used by several current planning systems is causal links, which are used to track search progress. The formalism combining Hierarchical Task Network (HTN) planning with these links known from Partial Order Causal Link (POCL) planning is often referred to as hybrid planning. In this paper we study the computational complexity of such hybrid planning problems. More specifically, we provide missing membership results to existing hardness proofs and thereby provide tight complexity bounds for all known subclasses of hierarchical planning problems. We also re-visit and correct a result from the literature for plan verification showing that it remains NP-complete even in the absence of a task hierarchy.

1 Introduction

Hierarchical planning has attracted much attention in the last decade [Bercher *et al.*, 2019]. Although there are many different variants of hierarchical planning formalisms [Bercher *et al.*, 2016; 2019], the de-facto "standard" variant is referred to as Hierarchical Task Network (HTN) planning [Erol *et al.*, 1996; Bercher *et al.*, 2019]. Most of the progress of recent years in hierarchical planning – particularly with regard to theoretical investigations – was done for this HTN planning formalism [Bercher *et al.*, 2019]. Several systems capable of solving HTN planning problems are however capable of dealing with more complex problem descriptions that go beyond those of standard HTN planning. One such extension of HTN planning integrates causal links known from Partial Order Causal Link (POCL) planning [Penberthy and Weld, 1992; Weld, 1994] into the model and planning process – the resulting framework is sometimes referred to as *hybrid planning* [Kambhampati *et al.*, 1998; Biundo and Schattenberg, 2001; Biundo *et al.*, 2011; Bercher *et al.*, 2016; 2017].

Several recent hierarchical planners rely on a search in the space of partial plans that exploit causal links to track progress, such as one of the PANDA planners [Bercher *et al.*, 2014; 2017]; the temporal planner FAPE [Bit-Monnot *et al.*, 2016; 2020] the runner-up of the partial-order HTN track of the International Planning Competition (IPC) 2020 pyHiPOP [Bechon *et al.*, 2014; Lesire and Albore, 2021]; HATP [Sebastiani *et al.*, 2017; Lallement *et al.*, 2018] and CHIMP [Stock *et al.*, 2015], two recent hierarchical planners for robotics; as well as a planner for the generation of narratives [Winer and Young, 2017]. Several POCL-based hierarchical planning formalisms furthermore allow for the compound tasks to specify preconditions and effects, just like the primitive tasks (i.e., the actions known from classical planning) do [Yang, 1990; Young *et al.*, 1994; Kambhampati *et al.*, 1998; Biundo *et al.*, 2011; Bechon *et al.*, 2014; Lesire and Albore, 2021; Bercher *et al.*, 2016; 2017] so they can be producers or consumers of causal links as well.

Knowing about the computational complexity of such hybrid models is not just an important theoretical insight in general, but since each search node can be interpreted as a planning problem itself, theoretical insights can also serve as the basis for problem relaxations and hence heuristics for such planners. Since hybrid problems extend the capabilities of standard HTN problems (by means of preconditions and effects of compound tasks as well as causal links), these problems are at least as hard as HTN problems [Bercher *et al.*, 2016]. However, it is unclear whether the presence of causal links and the way they restrict possible decompositions makes these problems computationally harder, i.e., membership results were not provided. We here close this gap and conduct a comprehensive complexity analysis for hybrid planning problems by studying the computational complexity of all known special cases on the task hierarchy from the hierarchical planning literature [Erol *et al.*, 1996; Alford *et al.*, 2012; 2015a]. We also correct a result from the literature for hybrid plan verification and show that for primitive problems it's not in **P**, but **NP**-complete. A minor contribution is our formalization itself as we make some simplifications to the one by Bercher *et al.* [2016] that not only make it easier to grasp, but this also resolves issues with the previous one.

2 Background

2.1 Hybrid Planning Formalization

In *HTN planning* there are two kinds of tasks: *primitive tasks*, the actions known from classical (non-hierarchical) planning; and *compound tasks*, which represent abstractions of var-

ious other primitive and compound tasks, each collection is grouped together by a so-called decomposition method. Problems are described in terms of an initial collection of such tasks, and the goal is to find a refinement into an executable plan consisting of primitive actions.

What we refer to as *hybrid planning* in this paper is a straight-forward extension of standard HTN planning by two concepts known from POCL planning [Penberthy and Weld, 1992; Weld, 1994], namely causal links and enabling compound tasks to specify preconditions and effects. Causal links "protect" task preconditions and make sure that they don't become false in the interval over which the link spans (formally defined later). For our problem formalization we heavily base upon the hybrid planning formalism by Bercher *et al.* [2016], which in turn extends the HTN formalism by Geier and Bercher [2011], for which most complexity results were shown – e.g,. for plan existence [Geier and Bercher, 2011; Alford *et al.*, 2014; 2015a; 2015b], plan verification [Behnke *et al.*, 2015], model adaptations [Behnke *et al.*, 2016; Lin and Bercher, 2021], and reasoning about landmarks and compounds tasks' guaranteed (i.e., inferred) preconditions and effects [Höller and Bercher, 2021; Olz *et al.*, 2021]. There are also more expressive HTN formalisms that allow, for example, the expression of state constraints on decomposition methods [Erol *et al.*, 1996], external conditions for methods [Tsuneto *et al.*, 1998], and method preconditions [Nau *et al.*, 2003]. We don't expect that the inclusion of such features would change the presented results (other than complicating the proofs), but didn't look into this yet.

A set of facts F is used to describe state properties. The set of states S is given by 2^F. Actions known from classical planning are referred to as primitive tasks. Their preconditions and effects induce state transitions as usual (see below), but since plans are partially ordered, we use causal links to track which preconditions are satisfied, and by which action the respective precondition is provided. For reasons that will become clear later, we require all preconditions and effects to be labeled. The labeling is provided by a function δ that maps primitive and compound task names $N = N_p \cup N_c$ to pairs of label and fact. Formally, $\delta : N \to 2^{\mathbb{N}\times F} \times 2^{\mathbb{N}\times F} \times 2^{\mathbb{N}\times F}$. Let $p \in N_p \subseteq N$ be a primitive task name and $\delta(p) = (prec, add, del)$. Let $prec', add', del' \subseteq F$ refer to the fact component of each respective entry. Then, p is applicable in a state $s \in S$, if and only if $prec' \subseteq s$. If applicable in s, p leads to $s' = (s \setminus del') \cup add'$. Applicability of action *sequences* is defined in the usual way by repeated action application.

We can now define *plans* (which in the context of HTN planning would be referred to as *task networks*), the most central concept in hierarchical planning. Plans are partially ordered sets of primitive and compound tasks, augmented by causal links. Ordering constraints and causal links are defined upon a set of *plan steps* (arbitrary symbols/labels) rather than on the task names directly to enable tasks to occur multiple times in the same plan.

Definition 1 (Plan)**.** *A* plan P *over a set of task names* N *is a 4-tuple* $(PS, CL, \prec, \alpha)$*, where:*

- PS *is a finite (possibly empty) set of plan steps,*
- $CL \subseteq (\mathbb{N} \times PS) \times F \times (\mathbb{N} \times PS)$ *is a set of causal links. Let* $(\langle i, ps\rangle, f, \langle i', ps'\rangle) \in CL$*. Then,* $(i, f) \in add(\alpha(ps))$ *and* $(i', f) \in prec(\alpha(ps'))$*. We call* ps *the* producer*,* ps' *the* consumer*, and* $f \in F$ *the protected condition of that causal link.*
- $\prec \subseteq PS \times PS$ *is a strict partial order. We require* $(ps, ps') \in \prec$ *for all* $(\langle i, ps\rangle, f, \langle i', ps'\rangle) \in CL$*, where* $\alpha(ps)$ *and* $\alpha(ps')$ *are primitive.*
- $\alpha : PS \to N$ *labels every plan step with its task name.*

$\mathcal{P}_N$ *denotes the set of all plans over the task names* N*. A plan is primitive if* $\{\alpha(ps) \mid ps \in PS\} \subseteq N_p$*.*

We will define the semantics of causal links after we provided the formal problem definition. Our definition of hybrid planning problems significantly builds upon Def. 2 by Bercher *et al.* [2016], but makes two noteworthy changes: First, we restrict to positive action preconditions, which simplifies the formalism without sacrificing expressivity [Behnke *et al.*, 2020]. Second, we exploit our newly introduced precondition/effect labels in the definition of decomposition methods. This slightly extends the syntactic problem definition, but it will pay off by significantly simplifying the definition of task decomposition and we will gain beneficial properties as discussed later when defining decomposition.

Definition 2 (Hybrid Planning Problem)**.** *A* hybrid planning problem *is a 6-tuple* $\pi = (F, N_c, N_p, \delta, M, P^I)$*, where:*

- F *is a finite set of* facts*,*
- *Let* $N := N_c \cup N_p$ *with* $N_c \cap N_p = \emptyset$*, and:*
  - N_c *is a finite set of* compound task names*,*
  - N_p *is a finite set of* primitive task names*,*
  - $\{init, goal\} \subseteq N_p$ *denote two special primitive task names encoding the initial and goal state,*
- $\delta : N \to 2^{\mathbb{N}\times F} \times 2^{\mathbb{N}\times F} \times 2^{\mathbb{N}\times F}$ *is a function mapping task names to their labeled preconditions and effects. For convenience, we also write* δ_{prec}*,* δ_{add}*, and* δ_{del} *to refer to the three co-domain components of* δ*.*
- M *is a finite set of* decomposition methods*, each* $m \in M$ *having the form* $m = (c, CL_{\mapsto}, P)$*, with* $c \in N_c$*,* $P \in \mathcal{P}_{N\setminus\{init,goal\}}$ *a plan, and* $CL_{\mapsto} : \mathbb{N} \to \mathbb{N} \times PS$ *defining a mapping from* c*'s preconditions and effects to those of plan steps* PS *in* P*. We require* $CL_{\mapsto}$ *to contain exactly one entry for each precondition and effect of* c*.*
- $P^I = (PS^I, CL^I, \prec^I, \alpha^I) \in \mathcal{P}_N$*, is the* initial plan*. It contains two plan steps* $ps_I, ps_G \in PS^I$ *such that:*
  - $\alpha^I(ps_I) = init$ *and* $\alpha^I(ps_G) = goal$*, and*
  - $ps_I \prec ps_G$ *and for all* $ps \in PS^I$ *with* $ps \notin \{ps_I, ps_G\}$ *holds* $ps_I \prec ps \prec ps_G$*.*

Having the syntactic problem definition at hand, we need to define the set of solutions. In hierarchical planning this requires transforming the initial plan into a solution plan by adhering the available decomposition methods which are applied to refine compound tasks into more primitive courses of action. Since those tasks can be involved in causal links, we first provide all formal definitions involving them.

First, recall that causal links are elements from $(\mathbb{N}\times PS)\times F \times (\mathbb{N} \times PS)$. Normally, causal links only specify the protected condition as well as the two involved plan steps [Penberthy and Weld, 1992; Weld, 1994; Bercher, 2021]. We, however, also include a label for each step since compound tasks may use some effect multiple times to get inherited to

different subtasks. One could even define some effect both as add and delete effect for that reason. The specification of the protected condition therefore even becomes redundant because also preconditions and effects are labeled, so providing just the labels in causal links would be sufficient to identify their protected condition. We provide that condition anyway for the sake of readability.

Let's start with the case where a causal link spans between two primitive tasks. Let $P = (PS, CL, \prec, \alpha)$, $cl = (\langle i, ps\rangle, f, \langle i', ps'\rangle) \in CL$, and $\alpha(ps), \alpha(ps') \in N_p$. The link cl now ensures that f (with $(i, f) \in add(\alpha(ps))$ and $(i', f) \in prec(\alpha(ps))$) will hold from the state in which ps produces it until it is required by ps'. This will be ensured by the solution criteria (defined later) that prevents *causal threats*, i.e., each action with a delete effect f has to be ordered either before ps (*promotion*) or behind ps' (*demotion*).

Causal links can however also involve compound tasks. In contrast to primitive actions, the preconditions and effects of compound tasks just indicate that these facts are required or produced by some subtasks. This implies that some precondition might be required by *some* subtask, whereas another might be required by *another* – and thus in different states. Potentially two compound task preconditions or effects could even be mutex since the respective subtasks are not bound to be executed in the same state. This is in contrast to compound task preconditions or effects that are required to hold in the same state [Olz *et al.*, 2021], but this rather loose/weak semantics appears to be widely deployed in various hybrid formalisms [Bercher *et al.*, 2016]. The labels of compound tasks' preconditions and effects further allows the mapping $CL_{\mapsto}$ to specify where each compound task's precondition/effect originates from. These will be the plan steps to which the respective causal link will be inherited down.

Definition 3 (Decomposition)**.** *Let $P = (PS, CL, \prec, \alpha)$ be a plan and $ps \in PS$ with $\alpha(ps) = c$. Assume $m = (c, CL_{\mapsto}, P_m) \in M$ with $P_m = (PS_m, CL_m, \prec_m, \alpha_m)$. W.l.o.g. we can assume $PS \cap PS_m = \emptyset$. Now, m decomposes ps in P into $P' = (PS', CL', \prec', \alpha')$ if and only if:*

- $PS' = (PS \setminus \{ps\}) \cup PS_m$
- $CL' = (CL \setminus CL_1) \cup CL_m \cup CL_2 \cup CL_3$, *with*

$CL_1 = \{(\langle i, ps'\rangle, f, \langle j, ps''\rangle) \in CL \mid ps \in \{ps', ps''\}\}$
$CL_2 = \{(\langle i, ps'\rangle, f, \langle k, ps_m\rangle) \mid (\langle i, ps'\rangle, f, \langle j, ps\rangle) \in CL, CL_{\mapsto}(j) = (k, ps_m)\}$
$CL_3 = \{(\langle k, ps_m\rangle, f, \langle j, ps'\rangle) \mid (\langle i, ps\rangle, f, \langle j, ps'\rangle) \in CL, CL_{\mapsto}(i) = (k, ps_m)\}$

- $\prec' = (\prec_1 \cup \prec_m \cup \prec_2 \cup \prec_3)$ *with*

$\prec_1 = (\prec \setminus \{(ps', ps'') \in \prec \mid ps \in \{ps', ps''\}\})$
$\prec_2 = \{(ps', ps_m) \mid ps_m \in PS_m, (ps', ps) \in \prec\} \cup \{(ps_m, ps') \mid ps_m \in PS_m, (ps, ps') \in \prec\}$
$\prec_3 = \{(ps', ps_m) \mid (\langle i, ps'\rangle, f, \langle j, ps\rangle) \in CL, CL_{\mapsto}(j) = (k, ps_m), \alpha(ps_m) \in N_p\} \cup \{(ps_m, ps') \mid (\langle i, ps\rangle, f, \langle j, ps'\rangle) \in CL, CL_{\mapsto}(i) = (k, ps_m), \alpha(ps_m) \in N_p\}$

- $\alpha = (\alpha \setminus \{(ps, c)\}) \cup \alpha_m$

We write $P \to_D^* P'$ if there is a (possibly empty) sequence of decompositions refining P into P'. In that case we call P' reachable from P. Let P_c denote a plan containing just the compound task c. If $P_c \to_D^* P''$ and the task t is contained in P'', then we say that t is reachable from c.

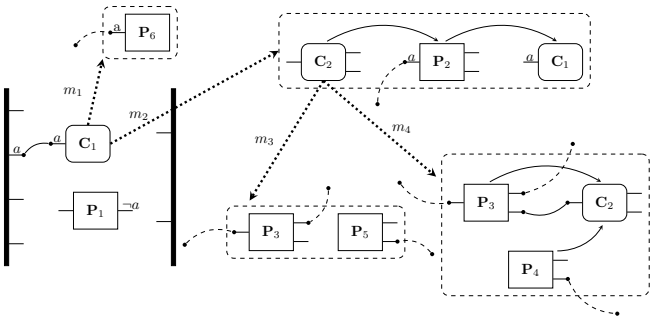

Figure 1: The left side depicts a plan plus the initial sate and goal description (shown as black vertical bars). Primitive tasks are depicted by boxes and use use $\mathbf{P}_i$ as a name, compounds tasks have rounded corners and are called $\mathbf{C}_i$. Plans of decomposition methods are shown in dashed boxes with round corners, dashed arrows show which methods are available for the respective compound tasks. Within each plan, solid arrows indicate ordering constraints, and solid edges causal links. Dashed edges within plans illustrate the respective method's map $CL_{\mapsto}$ by showing to which subtasks precondition or effect a causal link would get inherited should it be set.

This definition is a significant simplification of Def. 3 by Bercher *et al.* [2016] due to the fact that decomposition methods now specify how causal links get inherited down. According to the previous definition, each matching precondition of a subtask could be used for this purpose thus potentially producing up to an exponential number of successors in the search space for just a single decomposition. This could have happened when the decomposed compound task is involved in n causal links, and each could be inherited to m compatible subtask preconditions/effects, in which case there will be $O(n^m)$ successors. Our definition resolves this issue since each decomposition now results into exactly one successor node – as is also the case in HTN planning.

Example Before we move on defining the solution criteria we want to illustrate our definitions using Fig. 1. It provides an example for an initial plan, decomposition methods, and their mappings $CL_{\mapsto}$ (see the caption for explanations). On top of this, it also illustrates that even a single method can pose different constraints depending on the causal links its compound task is involved in (its impact therefore becomes context-sensitive). More specifically it shows that we are not allowed to only consider a method (i.e., its inheritance map) to decide upon the constraints it induces but rather we need to check whether the task it composes serves as producer or consumer of causal links. For example, consider the totally ordered method m_2 and assume for the sake of argument that all other methods and the initial plan were also totally ordered (we'll re-visit this figure where we'll need the partial order). Its map $CL_{\mapsto}$ shows that any link protecting $\mathbf{C}_1$'s precondition a will be inherited down to $\mathbf{P}_2$. First, note that in a total-order setting the producer of this a would come become before $\mathbf{C}_1$, so any such link would span over $\mathbf{C}_2$. Yet, this does not mean that we are allowed to remove all methods

from the subplans reachable from $\mathbf{C}_2$ from the model that delete a (e.g., in a preprocessing step) – because m_2 might also be used in a different plan where $\mathbf{C}_1$ will not have this causal link yet. This may be done for the instance of m_2 that's used to decompose the $\mathbf{C}_1$ that's part of m_2. ■

In practice, planning systems solving hybrid problems do not just perform task decomposition, but also ordering insertion and causal link insertion. Rather than defining these *explicitly* as operations on plans (as done in the literature) we can simply define solutions as "supersets" of plans resulting from decomposition thus further shortening and simplifying the formalization by Bercher *et al.* [2016].

Definition 4 (Solution)**.** *A plan $P = (PS, CL, \prec, \alpha)$ is a solution to a hybrid planning problem $\pi = (F, N_c, N_p, \delta, M, P^I)$ with $P^I = (PS^I, CL^I, \prec^I, \alpha^I)$ iff:*

1. *$P^I \rightarrow_D^* P' = (PS', CL', \prec', \alpha')$ and $PS = PS'$, $CL \supseteq CL'$, $\prec \supseteq \prec'$, and $\alpha' = \alpha$.*
2. *P is primitive and executable, the latter meaning:*
   - *each precondition is protected by a causal link, and*
   - *there are no* causal threats.

Criterion 1 comes from HTN planning, which requires that every solution is a refinement of the initial task network via applying the model's methods. There, such a task network is regarded a solution if it possesses an executable linearization. Since in hybrid planning causal links are the means to establish executability we need to ensure that missing causal links may be inserted as well as ordering constraints to resolve causal threats – both is ensured by allowing $CL \supseteq CL'$ and $\prec \supseteq \prec'$. Criterion 2 then demands the standard POCL solution criteria thus ensuring that every linearization is executable and making all goals true [Bercher, 2021].

2.2 Known Subclasses

HTN planning is known to be undecidable in general [Erol *et al.*, 1996; Geier and Bercher, 2011]. Both Erol *et al.* [1996] as well as Alford *et al.* [2012; 2015a] have identified various special cases that make the problem computationally easier. For the complexity investigations we conduct in the next section we investigate all subclasses known to make HTN planning easier. These restrict partial order or the task hierarchy, i.e., the interaction between compound tasks. We now review and replicate these definitions from the literature.

We start with totally ordered problems, which are known to be in **EXPTIME** [Erol *et al.*, 1996] as well as **EXPTIME**-hard [Alford *et al.*, 2015a]. This restriction can be posed in addition to other restrictions.

Definition 5 (Totally Ordered Problems)**.** *A hybrid planning problem is called* totally ordered *if the initial plan as well as all plans in all decomposition methods are totally ordered.*

Acyclic problems (also called non-recursive) are known to be decidable as well, since the search space is finite [Erol *et al.*, 1996]. They were later proved to be **NEXPTIME**-complete [Alford *et al.*, 2015a]. Acyclicity is also orthogonal to other properties and can thus also be be studied on top of other restrictions.

Definition 6 (Acyclic Problems)**.** *A hybrid planning problem is called* acyclic *(or* non-recursive*) if there is only a finite number of possible decompositions.*

The class of regular problems was also identified by Erol *et al.* [1996]. It is especially interesting as every classical planning problem can be described as a regular HTN problem (and vice versa). These problems are thus **PSPACE**-complete [Erol *et al.*, 1996].

Definition 7 (Regular Problems)**.** *A hybrid planning problem is called* regular *if the initial plan as well as all plans in all decomposition methods are regular. A plan is called regular if it has at most one compound task and if it does, it must be a "last task", i.e., all other tasks are ordered before it.*

Alford *et al.* [2012] found a generalization of regular problems, called *tail-recursive*. In tail-recursive problems, methods may contain more than one compound task, but arbitrary recursion is only allowed through the very last task, all others need to become "easier" upon decomposition. These problems are **EXPSPACE**-complete [Alford *et al.*, 2015a]. Formally, they are defined as follows:

Definition 8 (Tail-recursive Problems)**.** *First, we define a stratification on a set S is a total order $\leq$ on S. An inclusion-maximal subset $S' \subseteq S$ is a stratum if for all $x, y \in S'$ both $x \leq y$ and $y \leq x$ holds. We can now define a hybrid planning problem as* tail-recursive *if we can define a stratification $\leq$ on its tasks $N_p \cup N_c$, where for all methods $(c, CL_{\mapsto}, P)$ with $P = (PS, CL, \prec, \alpha)$ holds:*

- *if there exists a last compound plan step $ps \in PS$, we have $\alpha(ps) \leq c$, and*
- *for any non-last compound plan step $ps' \in PS$, we have $\alpha(ps') < c$ (i.e., $\alpha(ps') \leq c$ and $c \not\leq \alpha(ps)$).*

Tail-recursive problems are at the moment the computationally hardest subclass that is known to be decidable. As will be seen later, the way how compound tasks decrease in hardness (the level of their stratum), plans are bounded in size when these problems are solved with a progression-based search algorithm [Alford *et al.*, 2012].

Example Revisiting the domain and initial plan depicted in Fig. 1, we can make multiple observations. It's not totally ordered because neither are the initial plan nor method m_4. The problem isn't acyclic because both $\mathbf{C}_1$ and $\mathbf{C}_2$ decompose into themselves. The problem isn't regular because of the initial plan, but also because of m_2 which has more than one compound task. The problem is however tail-recursive, which we verify now. We need to be able to find a stratification that satisfies the requirement of the interaction between compound tasks and the methods' ordering constraints. We see that C_1 decomposes into C_2 but not vice versa which puts C_1 on a strictly higher stratum than C_2. The strata of primitive tasks essentially don't matter because one could just put all primitive tasks into their own and lowest stratum. Here however we distributed them differently (purely for illustrative and didactic purposes). The stratification we found is depicted in Fig. 2. We can verify that is satisfies all criteria from Def. 8 thus making the planning problem tail-recursive as we'll see now.

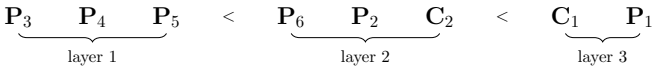

Figure 2: Graphical illustration of a stratification based on the planning problem depicted in Fig. 1.

Methods m_1 and m_3 could in principle be ignored since they only contain primitive tasks. But since we put them not into the lowest stratum we have to check them as well. m_1 decomposes $\mathbf{C}_1$, which is on layer 3. Since its sole task $\mathbf{P}_6$ is on layer 2, the constraints are satisfied. Likewise for m_3, which decomposes $\mathbf{C}_2$, which is on layer 2. Its tasks $\mathbf{P}_3$ and $\mathbf{P}_5$ are on layer 1 hence also satisfying all constraints.

Method m_2 is more interesting. Its first task is $\mathbf{C}_2$. Because it's a non-last task it must be on a lower stratum than the task decomposed (i.e., $\mathbf{C}_1$), which is the case. P_2 is primitive so it can be disregarded anyway, but according to the stratification we chose its stratum also satisfies all constraints as it also lies on a lower stratum than $\mathbf{C}_1$. Finally, $\mathbf{C}_1$ itself is also contained in m_2 that decomposes $\mathbf{C}_1$. This is however not a problem because it's the last task in that method and therefore allowed to be on the same stratum as the task the method decomposes, which is the case here.

Finally, m_4 also doesn't violate tail-recursion. $\mathbf{P}_3$ and $\mathbf{P}_4$ are primitive and hence not problematic, but we also see that their stratum (layer 1) is below that of $\mathbf{C}_2$, which is 2. The only compound task in that method is $\mathbf{C}_2$, but again because it's the last task it's allowed to be on the same level as the task it decomposes (i.e., also $\mathbf{C}_2$), so that again works. ■

3 Complexity Investigations

We consider both the the *plan existence problem* (is there a solution?) as well as *plan verification* (is the given plan a solution?). We start with the former.

3.1 Plan Existence Problem Complexity

The only complexity investigations for hybrid problems we are aware of are those by Bercher *et al.* [2016] who showed undecidability as well as semi-decidability (Thms. 4 and 5). Semi-decidability is clear since one can simply perform a breadth-first search. Undecidability may appear a trivial corollary as hybrid problems are a special case of HTN problems. However, Bercher *et al.* [2016] showed this in the presence of various so-called *legality criteria* (Defs. 7 to 10) which restrict the set of decomposition methods one is allowed to specify. One of their main results is however that any HTN problem can be turned into a hybrid problem that satisfies all investigated legality criteria (Thm. 1). This also shows that these criteria are not beneficial as none of them seems to provide any useful properties – which is why we do not consider them in this work.

The complexity for the general case was already shown in the literature. We repeat it here for the sake of completeness.

Theorem 1 (Thms. 4 and 5 by Bercher *et al.* [2016])**.** *Hybrid planning plan existence is undecidable and semi-decidable.*

Again, as every HTN problem is by definition also a hybrid problem, we furthermore get the following corollary:

Corollary 1. *Every hybrid problem of a certain subclass (e.g., total order etc.) is as hard as HTN problems of the respective class i.e., has the same lower bound.*

Note that the previous corollary holds for all possible subclasses, including those not yet discovered. Membership proofs will however have to be able to deal with causal links and the state constraints they induce. We investigate the complexities in the order we reviewed the respective restrictions.

We start our investigations with the complexity of regular problems and their extension to tail-recursive problems. Deciding them relies on *progression*, a common technique in HTN planning [Alford *et al.*, 2012] usually deployed in state-based HTN planners like SHOP [Nau *et al.*, 2003] or one of the PANDA planners [Höller *et al.*, 2018]. Although progression would usually not be applied in plan space-based planning, we are still *able* to use it in this context, and in fact doing so will provide us with matching upper bounds.

Progression identifies tasks in a task network that according to the ordering constraints could be executed next. Such steps are called unconstrained. Formally, a step $ps \in PS$ is *unconstrained* if $\forall ps' \notin \{ps_I, ps\} : (ps', ps) \notin \prec$, i.e., if it doesn't have any predecessors according to ordering constraints (except for the initial state step). If such an unconstrained task is compound it gets decomposed, if it's primitive it gets applied in the current state, updating it. This works in hybrid planning as well, we just need to keep track whether causal links cause unresolvable threats and to re-link causal links to the initial state after action application. Formally, progression in hybrid planning can be defined as follows.

Definition 9 (Progression)**.** *Let $\pi = (F, N_c, N_p, \delta, M, P)$ be a problem with $P = (PS, CL, \prec, \alpha)$ a plan and $ps \in PS$ an* unconstrained *plan step with $\alpha(ps) = t$. If $t \in N_c$ and m is a method for t that decomposes P into P', then $\pi' = (F, N_c, N_p, \delta, M, P')$ is a progression of π through ps.*

Otherwise, if t is primitive ($t \in N_p$), applicable in the initial state and does not threaten any causal links between the initial state and another task, then it can be progressed *into a new planning problem without t. Let the labels in $\delta(init)$ and $\delta(t)$ be disjoint. $\pi' = (F, N_c, N_p, \delta', M, P')$ with $P' = (PS', CL', \prec', \alpha')$ is a progression of P through ps if and only if:*

- $\delta'(x) = \delta(x)$ *for all* $x \neq init$
- $\delta'(init) = (\emptyset, (\delta_{add}(init) \setminus \delta_{del}(t)) \cup \delta_{add}(t), \emptyset)$
- $PS' = (PS \setminus \{ps\})$
- $CL' = (CL \setminus CL_1) \cup CL_2$, *with*

$CL_1 = \{(\langle i, ps' \rangle, f, \langle j, ps'' \rangle) \in CL \mid ps \in \{ps', ps''\}\}$
$CL_2 = \{(\langle k, ps_I \rangle, f, \langle j, ps' \rangle) \mid (\langle i, ps \rangle, f, \langle j, ps' \rangle) \in CL\}$
choosing k so that $(k, f) \in \delta'_{add}(init)$ iff $(i, f) \in \delta_{add}(t)$

- $\prec' = (\prec \setminus \{(ps', ps'') \in \prec \mid ps \in \{ps', ps''\}\})$
- $\alpha' = \alpha \setminus \{(ps, t)\}$

A sequence of progressions from problem π to π' is denoted as $\pi \rightarrow_P^* \pi'$.

Progression may increase a plan's size in case of decomposition (if the used method isn't empty) but decreases it when progressing a primitive task. Thus a problem is regarded solved if there's a sequence of progressions that turns the problem's initial plan into an empty one. Formally, if there

is a π'' with an empty plan, such that $\pi \rightarrow_P^* \pi''$, then π is solvable. We omit a formal proof for this, but it can easily be seen that the sequence of applied actions corresponds to an executable task linearization of a solution according to Def. 4, similar to HTN planning [Alford *et al.*, 2012].

Example We now revisit Fig. 1 but assume that the initial plan is just any current plan, i.e., the state depicted on the left is now the current state rather than the problem's initial state. In that plan, there are two tasks without predecessors, so both could potentially be chosen to produce a successor plan. Assume that $\mathbf{P}_1$'s precondition is satisfied in the current state. Despite that, it can't be "progressed" because its delete effect a would violate the causal link rooting in the current state. So instead we can only choose to refine $\mathbf{C}_1$ producing two successor plans, one for each available method. Say m_1 is chosen. In the resulting plan (where the current state's outgoing causal link would be re-linked to $\mathbf{P}_6$) only $\mathbf{P}_6$ can be progressed, after which finally $\mathbf{P}_1$ could be progressed as well. Note that in this progression algorithm there's no need to insert causal links to achieve executability as it's usually done in POCL planning as we still rely on the standard state-based criterion for checking executability. We only use causal links that are present in decomposition methods (and are that 'forced' into plans), but don't add new ones. ■

Theorem 2. *Regular as well as totally-ordered tail-recursive hybrid problems can be decided in **PSPACE**, partially ordered tail-recursive problems in **EXPSPACE**.*

Proof. We use progression to solve these problems.

Each decomposition may only add a finite number of causal links, but the total number of links per plan can be bounded by the available preconditions and effects in said plan. Given a plan's set of plan steps PS and its largest set of preconditions p and effects e, there are at most $|PS| \cdot |p| \cdot |e|$ non-redundant causal links, which is a polynomial.

For regular problems, each decomposed task must be the last, therefore all reachable plans are bounded by the size of the largest plan in the methods (plus potentially additional causal links that get propagated to the next plan, but this is still polynomially bounded as stated above). Running a non-deterministic progression search thus takes **NPSPACE**, which is **PSPACE** [Savitch, 1970].

Tail-recursive problems can also be decided using progression since the size of each plan under progression can be bounded [Alford *et al.*, 2015a]. This is because each task, if compound, adds at most one task with the same 'hardness', all others have to precede it in this case, and are strictly easier. So there is only a finite number of decompositions that may increase the size of any plan under progression. This maximal size of task networks/plans under progression is called progression bound [Alford *et al.*, 2015a], which we show to not increase in the hybrid setting.

For *partially-ordered* tail-recursive problems, the largest task network reachable under progression is exponential in size [Alford *et al.*, 2015a]. Combined with the bound on causal links, this puts partially-ordered tail-recursive hybrid planning in **NEXPSPACE** = **EXPSPACE**.

For totally-ordered tail-recursive problems, if s is the number of layers in a problem's (i.e., plan's) stratification and m is the largest method, there can be at most $s \cdot |m|$ tasks in any reachable problem [Alford *et al.*, 2015a]. This, with the polynomial number of causal links for a given plan, means there is a polynomial bound on the size of any reachable problem/plan, making it **NPSPACE** = **PSPACE** as well. □

Theorem 3. *Totally ordered hybrid problems can be decided in **EXPTIME**.*

Proof. Our proof is an extension of the one for Thm. 4 by Erol *et al.* [1996] for **EXPTIME**-membership of totally ordered HTN problems. We first recap their exact proof for HTN planning before extending it for hybrid problems (without increased complexity despite dealing with causal links).

Proof for HTN planning The proof is a dynamic programming procedure that builds a huge table of exponential size that checks for each combination of input state s, primitive or compound task task n, and successor state s' whether n can be transformed from s into s'. For a primitive task that's just a simple state transition, for a compound task that implies checking whether a primitive refinement with the respective property exists. More precisely, the table is the complete set $2^F \times (N_c \cup N_p) \times 2^F \times \{\top, \bot, \textit{unknown}\}$. Note that even a *complete* table has size $2^{|F|} \cdot |N_c| \cdot |N_p| \cdot 2^{|F|} \cdot 3$, which can be bounded by $O(2^{2|F|})$, which is an exponential. By relying on dynamic programming it can be shown that runtime doesn't exceed the construction process of this table thus resulting in an **EXPTIME**-membership proof.

We start by initializing the table with unknowns. Then we fill all entries of primitive tasks by simply applying them and checking the resulting state. In a next step, we identify all compound tasks that admit a decomposition method that contains primitive tasks only. Let $p_1, \ldots, p_n$ such a primitive task sequence of such a method. By assumption, we know that we have all table entries filled out for all these tasks and all combinations of predecessor and successor states. We therefore can now investigate every combination of initial and intermediate states as well. We thus check for each sequence $s_0, \ldots, s_n$ whether $p_1, \ldots, p_n$ can be applied to s_0 leading into s_n. For the method's compound task c and each pair of states s_0, s_n we update the table's *unknown* truth value accordingly to $\top$ if it's executable and keep it *unknown* otherwise as there might be other decomposition methods (processed later) that work. Note that now we have $(2^{|F|})^{(n+1)} = 2^{|F| \cdot (n+1)}$ many states to check, which is still an exponential (not a double-exponential). We continue with this procedure in a bottom-up fashion by investigating all methods for which all tasks (primitive or compound) are marked executable. No method has to be processed twice, so eventually we terminate and capture the initial task network. Technically, we have to compile the initial task network into a new method for some new compound task c_I so that we can look up the truth value for c_I. After the table update process terminated, the result is yes if and only if the entry for c_I is $\top$. This completes the proof (by Erol *et al.* [1996]) for the HTN setting.

Extensions to hybrid planning In hybrid planning, we also have to deal with the inheritance of causal links, the main problem being that even the same method might induce different constraints depending on the context in which it was used. Some task c with a precondition a might be contained in a plan twice, once with a link pointing to its precondition and once without. This means that even though each method has to specify to which subtask this condition a gets inherited down, the constraint for the respective causal link might not exist – because this depends on whether the task c was in fact involved in a causal link or not. This means that we cannot just check method feasibility for its compound task alone, but need to consider in which causal links it is involved in. To store whether links that span over a task are respected we also maintain a set of conditions dur (during) for each task. For this reason, we extend the table to: $2^F \times (N_c \cup N_p) \times (2^F)^4 \times \{\top, \bot, \mathit{unknown}\}$. Thus we added three entries before the truth value, each being a subset of the state variables. In order, they represent the conditions protected by an incoming link, those protected by an outgoing causal link, and those protected during the complete execution of the respective task (i.e., those protected by links spanning *over* the task). We will refer to those three sets as *in*, *out*, and *dur*, respectively. Whereas the *dur* set is relevant for *all* tasks (primitive or compound), the causal link conditions in in and out are only relevant for compound tasks as they serve the purpose of knowing whether a method's mapping will induce a constraint or not.[1] For primitive tasks, these entries serve no purpose as they don't constrain themselves. Note that the size of this table has increased to $O(2^{5|F|})$, but this is still an exponential.

The procedure works as before, with only minimal changes. For primitive tasks we need to check whether the protected conditions (in their *dur* set) don't change. If they do, their truth value has to be set $\bot$. For the bottom-up propagation step assume again that $P_m \equiv t_1, \ldots, t_n$ is a totally ordered task network in a method $m = (c, CL_{\mapsto}, P_m) \in M$ for some compound task c, such that we obtained all truth values in the table for the involved tasks. Recall that the entry for c that we investigate now also contains a complete specification of its ingoing protected preconditions in, outgoing protected effects out, and the conditions dur protected by links that span over c itself and thus all tasks in all its methods. We can thus exploit in, out, and $CL_{\mapsto}$ to find out to which subtasks causal links will be inherited down. Thus, if some precondition a of c is protected (i.e., given in the table entry) we know due to $CL_{\mapsto}$ to which task it is inherited down and thus know that the condition should be in the dur set of all preceding tasks. The same can be done for effects. We furthermore have to check that all causal links in the plan itself are being satisfied. This is done in exactly the same way by looking up the entries for which the respective dur set contains the respective value. Finally, each condition in the dur set of c must be in the dur sets of all sub tasks. With this change, we will again reach the initial plan eventually and look up whether it is executable thus completing the proof. □

[1]Note that the table and thus proof is a slight simplification here since it's possible to specify some compound task's precondition or effect multiple times, which would require to not just store the protected conditions, but also their number. This however doesn't influence complexity, so we omitted this for the sake of simplicity.

Theorem 4. *Acyclic hybrid problems are in **NEXPTIME**, acyclic totally ordered hybrid problems are in **PSPACE**, primitive problems are in **NP**, and primitive totally ordered problems are in **P**.*

Proof. We follow (parts of) the membership proof by Geier and Bercher [2011] for HTN problems with task insertion (TIHTN problems) [Geier and Bercher, 2011; Alford *et al.*, 2015b]. We can guess and verify: We guess a so-called decomposition tree [Geier and Bercher, 2011], which is simply a tree-representation of a sequence of decomposition methods thus leading to a certain plan (by applying those methods). Since the problem is acyclic, the tree's size can be bounded by an exponential in the problem size. We only need to verify that the tree is "compatible with the model", i.e., that it indeed represents an applicable sequence of methods. This can be done in polynomial time in the size of the tree [Geier and Bercher, 2011] (assuming we also guessed a mapping between the task labels in the tree and those in the methods, otherwise we also had to solve graph isomorphism problems, which might not be in **P**) and does not become harder in the hybrid setting since link inheritance is a trivial test according the the respective method's map. For executability the resulting plan might require additional ordering constraints and causal links, but those can be guessed as well in the size of the resulting plan (which is also exponentially long) and verified in polynomial time [Bercher, 2021] – giving us a **NEXPTIME** decision procedure in total.

Acyclic problems are by definition tail-recursive [Alford *et al.*, 2015a] because every decomposition makes all resulting tasks "easier" (i.e., decreases their stratification level). Therefore totally ordered acyclic problems are also totally ordered tail-recursive, which we proved to be in **PSPACE**.

Primitive problems can be regarded a special case of acyclic problems, as no decompositions are possible here at all. This leaves us with the last step of guessing orderings, causal links, and verifying – showing **NP** membership. If the plan is totally ordered already, then clearly we only need to check whether existing causal links cause threats (if so reject since they can't be resolved) and whether we can insert causal links to make it executable. The latter is yes if and only if the action sequence is executable [Bercher, 2021, Prop. 1]. Since executability can clearly be checked in polynomial time, **P** membership follows directly. □

Corollary 2. *The special cases of hybrid planning problems listed below all have the same computational complexity as their HTN variants. More precisely:*

- *Primitive: **NP**-complete* *[Erol* et al.*, 1996]*
  *& total order: **P*** *[no hardness shown]*
- *Total order: **EXPTIME**-complete [Alford* et al.*, 2015a]*
- *Acyclic: **NEXPTIME**-complete [Alford* et al.*, 2015a]*
  *& total order: **PSPACE**-complete [Alford* et al.*, 2015a]*
- *Regular: **PSPACE**-complete* *[Erol* et al.*, 1996]*
- *Tail-rec.: **EXPSPACE**-complete [Alford* et al.*, 2015a]*
  *& total order: **PSPACE**-complete [Alford* et al.*, 2015a]*

The previous corollary concludes tight complexity results for all known subclasses of hybrid planning problems. Shown citations indicate sources for the respective hardness results (applied to Cor. 1).

3.2 Plan Verification Complexity

Bercher *et al.* [2016] studied the hybrid plan verification problem and proved its **NP**-completeness for the general case.

They also studied the special case where not only the plan to verify is primitive, but also the planning problem itself. They stated this could be checked in polynomial time [Bercher *et al.*, 2016, Thm. 2], but their result is incorrect. The authors' proof just checks whether the plan to verify is executable – and nothing else. However for a plan to be a solution it must also be a refinement of the initial plan, which they did not check. For primitive plans this means checking whether ordering and link insertion can turn one plan into the other (cf. Def.4). This is apparently already **NP**-hard.

Definition 10 (Plan Verification Problem)**.** *Let π be a hybrid planning problem. The* hybrid plan verification problem *is to decide whether a given plan P is a solution to π.*

The intuition about why verifying whether P is a solution to π is **NP**-hard even for primitive plans is that the set PS of plan steps in P may be disjoint with that in P^I (note that they are just arbitrary labels). Hence, in order to determine whether P is a solution, we need to find whether there exists a bijective mapping between those two sets which preserve the structure of P^I. Moreover, since ordering constraint and causal link insertions are allowed in hybrid planning, it eventually leads us to solving a sub-graph isomorphism problem. Behnke *et al.* [2015] phrased this for HTN planning:

Definition 11 (Plan Compatibility Problem)**.** *Given two plans $P_1 = (PS_1, \emptyset, \prec_1, \alpha_1)$ and $P_2 = (PS_2, \emptyset, \prec_2, \alpha_2)$ with $|\prec_1| \leq |\prec_2|$, the plan compatibility problem is to decide whether P_1 is compatible with P_2, i.e., whether there exists a bijective mapping $\beta : PS_1 \to PS_2$ such that for every $ps \in PS_1$, $\alpha_1(ps) = \alpha_2(\beta(ps))$, and for every $ps, ps' \in PS_1$, if $(ps, ps') \in \prec_1$, then $(\beta(ps), \beta(ps')) \in \prec_2$.*

Behnke *et al.* [2015] proved that this problem is **NP**-complete (Thm. 3). By investigating their reduction for hardness one can see that they construct a plan where no action has any preconditions and effects thus making already this special case **NP**-hard. Notice that this problem is just a special case of plan verification for non-hierarchical hybrid problems, which we exploit for hardness of the latter problem.

Theorem 5. *The plan verification problem for primitive hybrid problems is **NP**-complete.*

Proof. Let $P = (PS, CL, \prec, \alpha)$ be a primitive plan and $\pi = (F, \emptyset, N_p, \delta, \emptyset, P^I)$ a primitive hybrid planning problem where $P^I = (PS^I, CL^I, \prec^I, \alpha^I)$.

For membership, we first guess a bijective mapping β from PS^I to PS and then, based upon the solution criteria of hybrid planning problems (Def. 4), verify whether the following criteria are satisfied: 1) For each $(\langle i, ps\rangle, f, \langle j, ps'\rangle) \in CL^I$, $(\langle i, \beta(ps)\rangle, f, \langle j, \beta(ps')\rangle) \in CL$, 2) for each $(ps, ps') \in \prec^I$, $(\beta(ps), \beta(ps')) \in \prec$, and 3) P is executable, that is, every precondition of every plan step in P is protected by a causal link, and there are no causal threats in P. Clearly, the first two criteria can be verified in time $\mathcal{O}(|CL|)$ and $\mathcal{O}(|\prec|)$, respectively. For the last one, we can iterate through all plan steps ps's in P and check 1) whether each precondition of ps is protected and 2) for every other plan steps in P, whether it violates the causal links protecting ps's precondition. Checking these two points takes time $\mathcal{O}(|PS|^2)$.

For hardness, we can reduce from the plan compatibility problem. Let P_1 and P_2 be the ones from Def. 11, i.e., both without causal links. We now construct a hybrid primitive plan verification problem with $\pi = (\emptyset, \emptyset, N_p, \delta, \emptyset, P^I)$, where δ maps all tasks to no-ops (which implies that all possible plans are executable). We set $P^I = P_1$ and the plan P to verify as P_2. We know that no plan may possibly contain causal links due to lacking preconditions. So $P = P_2$ is a solution to $P^I = P_1$ if and only if P_1 is compatible with P_2. Hardness follows directly since the the proof of Thm. 3 by Behnke *et al.* [2015] only considers the tasks with which plans steps are labelled, but doesn't consider their preconditions and effects. □

4 Conclusion

For hybrid planning, which combines HTN with POCL planning, only hardness results were known for the plan existence problem, upper bounds were missing. That is, it was not known whether causal links that may be part of the initial plan or any decomposition method – and the state constraints they induce – increase the computational hardness. By providing missing (complexity-matching) membership proofs we answered this question and thus provided tight complexity bounds for all known subclasses of hierarchical planning. We further investigated the plan verification problem and were able to correct a wrong result from the literature showing that verifying primitive solutions to non-hierarchical problems is already NP-hard. Another contribution of the paper is a simplified formalization of hybrid planning which also resolves issues with an existing formalization.

References

[Alford *et al.*, 2012] R. Alford, V. Shivashankar, U. Kuter, and D. Nau. HTN problem spaces: Structure, algorithms, termination. In *SoCS*, pages 2–9. AAAI Press, 2012.

[Alford *et al.*, 2014] R. Alford, V. Shivashankar, U. Kuter, and D. Nau. On the feasibility of planning graph style heuristics for HTN planning. In *ICAPS*, pages 2–10. AAAI Press, 2014.

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

[Alford *et al.*, 2015b] R. Alford, P. Bercher, and D. Aha. Tight bounds for HTN planning with task insertion. In *IJCAI*, pages 1502–1508. AAAI Press, 2015.

[Bechon *et al.*, 2014] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal. HiPOP: Hierarchical partial-order planning. In *STAIRS*, pages 51–60. IOS Press, 2014.

[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 Press, 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 Press, 2016.

[Behnke *et al.*, 2020] G. Behnke, D. Höller, A. Schmid, P. Bercher, and S. Biundo. On succinct groundings of HTN planning problems. In *AAAI*, pages 9775–9784. AAAI Press, 2020.

[Bercher *et al.*, 2014] P. Bercher, S. Keen, and S. Biundo. Hybrid planning heuristics based on task decomposition graphs. In *SoCS*, pages 35–43. AAAI Press, 2014.

[Bercher *et al.*, 2016] P. Bercher, D. Höller, G. Behnke, and S.Biundo. More than a name? on implications of preconditions and effects of compound HTN planning tasks. In *ECAI*, pages 225–233. IOS Press, 2016.

[Bercher *et al.*, 2017] P. Bercher, G. Behnke, D. Höller, and S. Biundo. An admissible HTN planning heuristic. In *IJCAI*, pages 480–488. IJCAI, 2017.

[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.

[Bercher, 2021] P. Bercher. A closer look at causal links: Complexity results for delete-relaxation in partial order causal link (POCL) planning. In *ICAPS*, pages 36–45. AAAI Press, 2021.

[Bit-Monnot *et al.*, 2016] A. Bit-Monnot, D. E. Smith, and M. Do. Delete-free reachability analysis for temporal and hierarchical planning. In *ECAI*, pages 1698–1699. IOS Press, 2016.

[Bit-Monnot *et al.*, 2020] A. Bit-Monnot, M. Ghallab, F. Ingrand, and D. E. Smith. FAPE: a constraint-based planner for generative and hierarchical temporal planning, 2020.

[Biundo and Schattenberg, 2001] S. Biundo and B. Schattenberg. From abstract crisis to concrete relief – a preliminary report on combining state abstraction and HTN planning. In *ECP 2001*, pages 157–168. AAAI, 2001.

[Biundo *et al.*, 2011] S. Biundo, P. Bercher, T. Geier, F. Müller, and B. Schattenberg. Advanced user assistance based on AI planning. *Cognitive Systems Research*, 12(3-4):219–236, 2011.

[Erol *et al.*, 1996] K. Erol, J. A. Hendler, and D. S. Nau. Complexity results for HTN planning. *AMAI*, 18(1):69–93, 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 Press, 2011.

[Höller and Bercher, 2021] D. Höller and P. Bercher. Landmark generation in HTN planning. In *AAAI*, pages 11826–11834. AAAI Press, 2021.

[Höller *et al.*, 2018] D. Höller, P. Bercher, G. Behnke, and S. Biundo. A generic method to guide HTN progression search with classical heuristics. In *ICAPS*, pages 114–122. AAAI Press, 2018.

[Kambhampati *et al.*, 1998] S. Kambhampati, A.Mali, and B. Srivastava. Hybrid planning for partially hierarchical domains. In *AAAI*, pages 882–888. AAAI Press, 1998.

[Lallement *et al.*, 2018] R. Lallement, L. de Silva, and R. Alami. HATP: Hierarchical agent-based task planner. In *AAMAS*, pages 1823–1825. IFAAMAS, 2018.

[Lesire and Albore, 2021] C. Lesire and A. Albore. pyHiPOP – Hierarchical partial-order planner. In *IPC 2020 Booklet*, pages 13–16, 2021.

[Lin and Bercher, 2021] S. Lin and P. Bercher. Change the world – how hard can that be? on the computational complexity of fixing planning models. In *IJCAI*, pages 4152–4159. IJCAI, 2021.

[Nau *et al.*, 2003] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *JAIR*, 20:379–404, 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*, pages 11903–11912. AAAI Press, 2021.

[Penberthy and Weld, 1992] J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *KR*, pages 103–114. Morgan Kaufmann, 1992.

[Savitch, 1970] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sebastiani *et al.*, 2017] E. Sebastiani, R. Lallement, R. Alami, and L. Iocchi. Dealing with on-line human-robot negotiations in hierarchical agent-based task planner. In *ICAPS*, pages 549–557. AAAI Press, 2017.

[Stock *et al.*, 2015] S. Stock, M. Mansouri, F. Pecora, and J. Hertzberg. Online task merging with a hierarchical hybrid task planner for mobile service robots. In *IROS*, pages 6459–6464. IEEE, 2015.

[Tsuneto *et al.*, 1998] R. Tsuneto, J. Hendler, and D. Nau. Analyzing external conditions toimprove the efficiency of htn planning. In *AAAI*, pages 913–920. AAAI Press, 1998.

[Weld, 1994] D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[Winer and Young, 2017] D. R. Winer and R. M. Young. Merits of hierarchical story and discourse planning with merged languages. In *AIIDE*, pages 262–268. AAAI Press, 2017.

[Yang, 1990] Q. Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6(1):12–24, 1990.

[Young *et al.*, 1994] R. M. Young, M. E. Pollack, and J. D. Moore. Decomposition and causality in partial-order planning. In *AIPS*, pages 188–193. AAAI Press, 1994.