Barely Decidable Fragments of HTN Planning

P. Maurice Dekker,
Gregor Behnke
University of Amsterdam
p.m.dekker@uva.nl, g.behnke@uva.nl

Abstract

Unrestricted HTN planning is undecidable. However some fragments of HTN planning – such as totally-ordered or tail-recursive HTNs – are decidable. Studying these restricted fragments has lead to valuable insights, which ultimately gave rise to the development of new efficient HTN planning algorithms.

We identify new decidable fragments of HTN planning. In a *one-hole-digging* problem, every task network contains at most a single compound task. In an *initial* problem, compound tasks are order-minimal in all task networks, while in a *final* problem, they are order-maximal. We precisely determine the complexity of these fragments – they are Ackermann-complete. This remains true even under the restriction that there is only one compound task and only two methods for it.

Introduction

Hierarchical Task Network (HTN) planning (Sacerdoti 1975; Erol, Hendler, and Nau 1994) is an expressive formalism for planning. It allows for describing the physics of the domain in terms of both the preconditions and effects of actions but also allows for specifying a grammar-like refinement-structure that valid plans must follow. HTN planning has over the past decades been studied both from theoretical and practical views. In several cases, theoretical insights have informed practical planning methods and have lead to new algorithms. This includes for example Erol's insight that totally-ordered HTN planning is decidable (Erol, Hendler, and Nau 1996), which has lead to dedicated total-order HTN planners (Nau et al. 1999; Magnaguagno, Meneguzzi, and de Silva 2021; Schreiber 2021; Behnke, Höller, and Biundo 2018; Alford et al. 2016; Alford, Kuter, and Nau 2009), but also the complexity analysis of tail-recursive HTN problems (Alford, Bercher, and Aha 2015) and the idea of relating HTN planning to formal languages (Höller et al. 2014), which have lead to dedicated planning algorithms (Alford et al. (2016) and Höller (2021) respectively).

These innovations were, at least to some degree, driven by researchers wanting to better understand the complexity-landscape of HTN planning. As HTN planning in general is undecidable (Erol, Hendler, and Nau 1996), this at least includes identifying expressive but still decidable fragments of HTN planning. This had lead to discovering, e.g., totally-ordered and tail-recursive HTN problems.

We present a new class of decidable HTN planning problems: that of *one-hole-digging* problems. This class is orthogonal to the previously investigated classes and thus illuminates a new island of decidability. While being decidable, one-hole-digging problems are only barely decidable: we show that they are complete for the class of $\mathscr{F}_\omega$, i.e., for all problems whose runtime is limited by an Ackermann-function. Our completeness proof also goes through for other fragments of HTN planning, most notably *initial*, *final*, and *clean* problems. See Table 3 at the end of the paper.

Preliminaries

In order to present the new fragments and results, we first formally define the notion of HTN planning. We then introduce Petri nets, which we will use in our reduction, and lastly introduce the required concepts of computational complexity theory.

HTN Planning

In this section, we set up the HTN formalism following Geier and Bercher (2011). This is a simple way of adding hierarchy to the STRIPS formalism, and is hence comfortable for proving complexity results.

A *task network* over a set N of *task names* is a triple $\mathfrak{t} = (T, \prec, \alpha)$ where

- T is a finite set of *tasks*;
- $\prec$ is a strict partial order on T;
- $\alpha : T \to N$ assigns a task name to each task in the network.

Let $\mathfrak{T}_N$ be the set of all task networks over N. If $T' \subseteq T$, we define the *task subnetwork*

$$\mathfrak{t}|T' = \big(T', \prec \cap (T' \times T'), \alpha|T'\big) \in \mathfrak{T}_N.$$

For $t \in T$ we write $T \setminus t = T \setminus \{t\}$. We write $\mathfrak{t} \setminus t$ as a shorthand for

$$\mathfrak{t}|(T \setminus t).$$

An *embedding* $\phi : \mathfrak{t} \hookrightarrow \mathfrak{t}'$ of task networks $\mathfrak{t} = (T, \prec, \alpha)$ and $\mathfrak{t}' = (T', \prec', \alpha')$ is an injection $\phi : T \hookrightarrow T'$ that preserves the partial order both ways and satisfies $\alpha' \circ \phi = \alpha$. An *isomorphism* is a bijective embedding. Let $\mathfrak{o} = (\emptyset, \emptyset, \emptyset)$ be the empty task network. The symbol $\sqcup$ denotes the union of disjoint sets. A task network $\mathfrak{t} = (T, \prec, \alpha)$ is a *disjoint union* of a family $\{\mathfrak{t}_i : i \in I\}$ of task networks if there exist embeddings $\phi_i : \mathfrak{t}_i \hookrightarrow \mathfrak{t}$ such that

$$T = \bigsqcup_{i \in I} \operatorname{Im} \phi_i$$

and $\phi_i(t_i) \not\prec \phi_j(t_j)$ whenever $i, j \in I$ are distinct.

An *HTN problem* is a tuple $\Pi = (F, C, O, M, \delta, \mathfrak{t}_0, s_0)$ where

- F is a finite set of *(propositional) variables*;
- C is a finite set of *compound task names*;
- O is a finite set of *primitive task names*;
- $M \subseteq C \times \mathfrak{T}_{C \sqcup O}$ is a finite set of *(decomposition) methods*;
- $\delta : O \to \mathscr{P}(F)^4$ is an *action mapping*;[1]
- $\mathfrak{t}_0 \in \mathfrak{T}_{C \sqcup O}$ is an *initial task network*;
- $s_0 \subseteq F$ is an *initial propositional state*.

A *propositional state* of Π is a subset of F. For better readability, we will adopt the following style guide: facts are typewriter `blue`, compound task names in bold and **brown**, primitive task names in sans serif pink and decomposition methods in green. Let $\mathfrak{t} = (T, \prec, \alpha)$ be a task network over $C \sqcup O$. It is called a task network *of* Π if either $\mathfrak{t} = \mathfrak{t}_0$ or $(\mathbf{c}, \mathfrak{t}) \in M$ for some $\mathbf{c} \in C$. We call a task $t \in T$ *compound* if $\alpha(t) \in C$ and *primitive* if $\alpha(t) \in O$. We call $\mathfrak{t}$ *primitive* if all tasks in T are primitive (i.e. $\mathfrak{t}$ is a task network over O).

Applying a decomposition method changes a non-primitive task network into another task network. Suppose that $\mathfrak{t}_1 = (T_1, \prec_1, \alpha_1)$, $\mathfrak{t}_2 = (T_2, \prec_2, \alpha_2)$ and $\mathfrak{t} = (T, \prec, \alpha)$ are task networks, $t \in T_1$ and $\mu = (\alpha_1(t), \mathfrak{t})$ is a method. We write $\mathfrak{t}_1 \to_{t/\mu} \mathfrak{t}_2$ provided there exists an embedding $\phi : \mathfrak{t} \hookrightarrow \mathfrak{t}_2$ such that

$$T_2 = (T_1 \setminus t) \sqcup \operatorname{Im} \phi$$

and for all $t_1 \in T_1 \setminus t$ and $t' \in T$ and $* \in \{\prec, \succ\}$ it holds

$$t_1 *_1 t \Leftrightarrow t_1 *_2 \phi(t').$$

Intuitively this means that t got replaced by $\mathfrak{t}$. The task network $\mathfrak{t}_2$ exists and is unique up to isomorphism.

The *search space* of the problem Π is

$$\Omega_\Pi = \mathfrak{T}_{C \sqcup O} \times \mathscr{P}(F).$$

If $\mathsf{pr} \in O$ with $\delta(\mathsf{pr}) = (\pi_+, \pi_-, e_+, e_-)$ and $s \subseteq F$ satisfies $\pi_+ \subseteq s$ and $\pi_- \cap s = \emptyset$, we say pr is *applicable* in s and define $\gamma(s, \mathsf{pr}) = s \cup e_+ \setminus e_-$. Let $(\mathfrak{t}, s) \in \Omega_\Pi$ be an HTN state with $\mathfrak{t} = (T, \prec, \alpha)$ and let $t \in T$ be a primitive $\prec$-minimal task such that $\alpha(t)$ is applicable in s. Then write

$$(\mathfrak{t}, s) \rightsquigarrow_\Pi \Big(\mathfrak{t} \setminus t, \gamma\big(s, \alpha(t)\big)\Big) \in \Omega_\Pi.$$

If $\mathfrak{t}_1 \to_{t/\mu} \mathfrak{t}_2$ for some t and $\mu \in M$, also let

$$(\mathfrak{t}_1, s) \rightsquigarrow_\Pi (\mathfrak{t}_2, s).$$

[1]We include negative preconditions, but it is well-known that these can be compiled away in linear time; cf. (Gazen and Knoblock 1997, section 2.6).

The *search graph* of Π is $\Phi_\Pi = \big(\Omega_\Pi, \rightsquigarrow_\Pi, (\mathfrak{t}_0, s_0)\big)$, where $(\mathfrak{t}_0, s_0)$ is the initial HTN state. For the relation with other views on HTN search, we refer to (Alford et al. 2012). The problem Π is *solvable* if for some propositional state s_ω, the state $(\mathfrak{o}, s_\omega)$ is reachable in Φ_Π. If it is, it can be reached by first applying only decomposition methods until the task network is primitive, and then applying only actions.

If $\mathscr{Q}$ is a class of HTN problems, let PLANEX($\mathscr{Q}$) be the problem of deciding whether or not a given member of $\mathscr{Q}$ is solvable. This problem has been studied in the literature for various classes $\mathscr{Q}$. The complexities range from polynomial time to undecidable ((Erol, Hendler, and Nau 1994), (Geier and Bercher 2011), (Alford 2013), (Alford, Bercher, and Aha 2015)).

Petri Nets

A *Petri net* is a pair $\mathscr{N} = (P, \Theta)$ where P is a finite set of *places* and Θ is a finite set of *transitions*, which are functions $P \to \mathbb{Z}$. The Petri net $\mathscr{N}$ is called *ordinary* if $|\theta(p)| \leq 1$ for all $\theta \in \Theta$ and $p \in P$. Ordinary Petri nets have the same modelling power as arbitrary Petri nets (see (Murata 1989, section IV.A)). The *search space* $\Omega_\mathscr{N}$ of $\mathscr{N}$ is the set of all functions $P \to \mathbb{N}$. We work with pointwise addition of functions $P \to \mathbb{Z}$. For $\sigma, \sigma' \in \Omega_\mathscr{N}$, we define $\sigma \rightsquigarrow_\mathscr{N} \sigma'$ iff there exists a transition $\theta \in \Theta$ such that $\sigma + \theta = \sigma'$. This is called a *firing* of θ. Let $\Phi_\mathscr{N} = (\Omega_\mathscr{N}, \rightsquigarrow_\mathscr{N})$ be the *search graph*. A state $\sigma \in \Omega_\mathscr{N}$ is called a *unit state* of $\mathscr{N}$ if $\sigma(p) \leq 1$ for all $p \in P$.

Petri nets are often imagined to include an unlimited pool of *tokens*. State σ encodes that there are $\sigma(p)$ tokens at each place $p \in P$.

PETRI is the problem of deciding given an ordinary Petri net $\mathscr{N}$ and unit states τ_0, τ_1 of $\mathscr{N}$ whether or not τ_1 can be reached from τ_0 in $\Phi_\mathscr{N}$.

PETRIGEN is the same problem but without the "ordinary" and "unit" assumptions. For more information on this and similar problems we refer to (Esparza 1998). It is easy to see that PETRIGEN reduces to PETRI in polynomial time. Moreover, the following is standard:

Lemma 1. *Given a finite set* P *of instances of PETRI we can compute in polynomial time another instance of PETRI that has answer "yes" iff some instance in* P *has answer "yes".*

Complexity Theory

In this paper, we will consider complexity classes that lie beyond the typically considered hierarchy of complexity classes starting with $\mathbb{L}, \mathbb{NL}, \mathbb{P}, \mathbb{NP}, \mathbb{PSPACE}, \mathbb{EXP}, \ldots$. We introduce the notion of Ackermann-completeness following (Schmitz 2016).

We define $F_0(n) = n + 2$ and

$$F_k(n) = F_{k-1}^n(k) = \underbrace{F_{k-1}(\ldots(F_{k-1}}_{n \text{ time}}(k))\ldots).$$

(We pass k as an argument here to ensure that $\lim_{k\to\infty} F_k(0) = \infty$.) As a result, we get that $F_1(n) \geq 2n$, $F_2(n) \geq 2^n$ and

$$F_3(n) \geq \underbrace{2^{2^{\cdots^2}}}_{n \text{ high}}.$$

Lastly we define $F_\omega(n) = F_n(n)$, which is the Ackermann function. Let $\mathbb{ACKERMANN}$ be the class of all decision problems that can be solved by a deterministic Turing machine with a time bound of $F_\omega\big(F_k(n)\big)$ for an input length n for some $k \geq 1$. A problem solvable by a program with runtime $F_3(n)$ for an input length n, already need not be in the class ELEMENTARY which contains all problems solvable with run-time limited to some fixed height exponentiation tower. Problems in $\mathbb{ACKERMANN}$ can be still much harder.

A problem Π is $\mathbb{ACKERMANN}$-hard if for every problem $\Pi' \in \mathbb{ACKERMANN}$ there exists $k \in \mathbb{N}$ such that there exists a program that reduces any instance of Π' of some size n to an instance of Π in time at most $F_k(n)$. The intuition behind this definition is that any function F_k is negligibly small in comparison to the Ackermann function F_ω in the limit. As we can choose $k = 2$, it follows that exponential time reductions are valid for proving $\mathbb{ACKERMANN}$-hardness.

Leroux and Schmitz (2019) proved that PETRI $\in \mathbb{ACKERMANN}$. Recently, Czerwiński and Orlikowski complemented this result with hardness:

Theorem 2. *(Czerwiński and Orlikowski 2022) PETRI is* $\mathbb{ACKERMANN}$*-complete.*

New Fragments

Let $\Pi = (F, C, O, M, \delta, \mathfrak{t}_0, s_0)$ be an HTN problem and $\mathfrak{t}_0 = (T_0, \prec_0, \alpha_0)$.

- Π is *initial* if any compound task in any task network of Π is minimal w.r.t. the task network's order:
$$\forall \mathfrak{t} = (T, \prec, \alpha) : \big(\exists \mathbf{c} : (\mathbf{c}, \mathfrak{t}) \in M\big) \text{ or } \mathfrak{t}_0 = \mathfrak{t}$$
$$\implies \forall t \prec t' : \alpha(t) \in O.$$
- Π is *final* if any compound task in any task network of Π is order-maximal.
- Π is *clean* if it is initial and final.
- Π is *one-hole-digging* if every (initial or method) task network contains at most one compound task:
$$\left|\alpha_0^{-1}[C]\right| \leq 1 \;\&\; \forall\big(\mathbf{c}, (T, \prec, \alpha)\big) \in M : \left|\alpha^{-1}[C]\right| \leq 1.$$
- Π is *bottomless* if every primitive method task network is empty:
$$\forall(\mathbf{c}, \mathfrak{t}) \in M : \mathfrak{t} \in \mathfrak{T}_O \implies \mathfrak{t} = \mathfrak{o}.$$
- Π is *loop-unrolling*, if it only contains one compound task name and two methods:
$$|C| \leq 1 \;\&\; |M| \leq 2.$$

Note that a problem is clean iff compound tasks are isolated.

For a one-hole-digging problem, decomposition is a single sequence of methods that are successively applied to the only present compound task, and the number of compound tasks never exceeds 1 while moving through Φ_Π.

If some loop-unrolling Π with at least one compound task in the initial task network $\mathfrak{t}_0$ is solvable, there can only be one method – call it cont – with a non-primitive task network. The other method – call it stop – serves to end recursion. Consider some subcases:

- If Π is additionally one-hole-digging, the only choice to make for the decomposition of Π is how often to apply cont before applying stop.
- If Π is additionally bottomless, then
$$M = \big\{\text{cont} = (\mathbf{comp}, \mathfrak{t}), \text{stop} = (\mathbf{comp}, \mathfrak{o})\big\} \tag{1}$$
for some **comp** and $\mathfrak{t}$. If Π is additionally clean, let $\mathfrak{t}^{\mathrm{pr}}_{(0)}$ be the largest primitive task subnetwork of $\mathfrak{t}_{(0)}$; then the primitive task networks obtainable from $\mathfrak{t}_0$ with the decomposition methods in M are precisely the disjoint unions of $\mathfrak{t}^{\mathrm{pr}}_0$ with any number of copies of $\mathfrak{t}^{\mathrm{pr}}$.

Example 3. *Imagine we want to bury an object. The procedure is to dig a hole in the ground, put the object in it, and then cover it with the dirt that was dug up. The hole can be of any depth. Let* $F = \{\texttt{hole}, \texttt{buried}\}$, $C = \{\mathbf{bury}\}$, $O = \{\mathsf{dig}, \mathsf{put}, \mathsf{cover}\}$,
$$M = \big\{\mathit{deeper} = (\mathbf{bury}, \mathfrak{t}_{\mathit{deeper}}), \mathit{bottom} = (\mathbf{bury}, \mathfrak{t}_{\mathit{bottom}})\big\}$$
where $\mathfrak{t}_{\mathit{deeper}}$ *is the task network consisting of three tasks* $t^{\mathsf{dig}}_{\mathit{deeper}} \prec t^{\mathit{bury}}_{\mathit{deeper}} \prec t^{\mathsf{cover}}_{\mathit{deeper}}$ *with names* $\alpha_{\mathit{deeper}}(t^{\mathsf{dig}}_{\mathit{deeper}}) = \mathsf{dig}$, $\alpha_{\mathit{deeper}}(t^{\mathit{bury}}_{\mathit{deeper}}) = \mathbf{bury}$ *and* $\alpha_{\mathit{deeper}}(t^{\mathsf{cover}}_{\mathit{deeper}}) = \mathsf{cover}$, $\mathfrak{t}_{\mathit{bottom}}$ *is the task network consisting of one task* t_{bottom} *with name* $\alpha_{\mathit{bottom}}(t_{\mathit{bottom}}) = \mathsf{put}$, $\delta(\mathsf{dig}) = \langle \emptyset, \emptyset, \{\texttt{hole}\}, \emptyset\rangle$, $\delta(\mathsf{put}) = \big\langle \{\texttt{hole}\}, \emptyset, \{\texttt{buried}\}, \emptyset\big\rangle$, $\delta(\mathsf{cover}) = \langle \emptyset, \emptyset, \emptyset, \emptyset\rangle$, $\mathfrak{t}_0$ *is the task network consisting of one task* t_0 *with name* $\alpha_0(t_0) = \mathbf{bury}$ *and* $s_0 = \emptyset$*. Then* $\Pi = (F, C, O, M, \delta, \mathfrak{t}_0, s_0)$ *is a one-hole-digging loop-unrolling problem. Every path of* Φ_Π *leads to the goal, except when one immediately applies the* bottom *method or when one applies the* deeper *method forever.*

Let $\mathscr{I}$ be the class of initial problems. Let $\mathscr{F}$ be the class of final problems. Let $\mathscr{C}$ be the class of clean problems. Let $\mathscr{H}_1$ be the class of one-hole-digging problems. Let $\mathscr{B}$ be the class of bottomless problems. Let $\mathscr{L}$ be the class of loop-unrolling problems.

All *regular* problems, introduced by (Erol, Hendler, and Nau 1994), are final and one-hole-digging. However, not all final one-hole-digging problems are regular (there can be multiple maximal tasks). Still, if all task networks of some final problem are totally ordered, it is regular.

We call an HTN problem Π *quasi-final* if removing all maximal tasks from all task networks of Π results in an *acyclic* problem (see again (Erol, Hendler, and Nau 1994)). Then the remaining compound tasks can be compiled away in exponential time; thus we can compute a final problem that is solvable iff Π is solvable. Let $\mathscr{F}'$ be the class of quasi-final problems. Then it follows from the results below that PLANEX$(\mathscr{F}') \in \mathbb{ACKERMANN}$. Similar remarks hold for initial problems. Out of the IPC 2023, the domains AssemblyHierarchical, Blocksworld-HPDDL, Multiarm-Blocksworld, Robot, and Tower were quasi-final. However, as the task networks of these examples are totally ordered, they are actually in $\mathbb{EXP}$ (Erol, Hendler, and Nau 1996), so much easier than Ackermann.

Results

Intuitively, the class $\mathscr{H}_1$ might be complexity-wise close to other HTN classes or even classical planning as the restrictions to the allowed decompositions are severe. A method application either seems to "move the problem" by replacing the compound task with a new compound task and some primitive tasks, or readily creates a primitive task network. Example 3 is trivial, but this is largely caused by the total order in the method task network $\mathfrak{t}_{\text{deeper}}$. We will prove that the complexity with partial order is high: the problem PLANEX($\mathscr{H}_1$) is $\mathbb{ACKERMANN}$-complete. It is thus significantly more complex than any other known decidable HTN class and only barely easier than undecidable problems. The proof heavily relies on Theorem 2, by showing equivalence of PLANEX($\mathscr{H}_1$) to PETRI. As a bonus, our reductions work for $\mathscr{I}$ and $\mathscr{F}$ as well.

Membership

The idea of the membership proof is to use bi-directional search, a common technique in planning that already exists since (Pohl 1969). If $\Pi = (F,C,O,M,\delta,\mathfrak{t}_0,s_0)$ is an HTN problem, its *bi-directional search space* is

$$\Omega_\Pi^{\text{bi}} = \mathfrak{T}_{C\sqcup O} \times \mathscr{P}(F)^2.$$

The first propositional state in a triple in Ω_Π^{bi} is understood as the propositional state in forward search and the second in backward search. The search graph $\Phi_\Pi^{\text{bi}} = (\Omega_\Pi^{\text{bi}}, \rightsquigarrow_\Pi^{\text{bi}})$ over the bi-directional search space inherits the arrows $\rightsquigarrow_\Pi$ (where the propositional state in backward search does not change) with the addition of regression:

$$\Big(\mathfrak{t}, s, \gamma(s_1, \alpha(t))\Big) \rightsquigarrow_\Pi^{\text{bi}} (\mathfrak{t} \setminus t, s, s_1)$$

whenever $\mathfrak{t} = (T, \prec, \alpha)$ and $t \in T$ is $\prec$-maximal. It is easy to see that Π is solvable iff $(\mathfrak{o}, s, s)$ can be reached from $(\mathfrak{t}_0, s_0, s_\omega)$ in Φ_Π^{bi} for some propositional states s, s_ω.

If $\Pi \in \mathscr{H}_1$, we also inherit the property that the number of compound tasks remains at most 1 while moving through Φ_Π^{bi}.

Proposition 4. *PLANEX($\mathscr{H}_1 \cup \mathscr{I} \cup \mathscr{F}$) reduces to PETRI in exponential time.*

Proof. For each $\mathscr{Q} \in \{\mathscr{H}_1, \mathscr{I}, \mathscr{F}\}$ it is polynomial to decide whether a given HTN problem is in $\mathscr{Q}$. Hence it suffices to show for each such $\mathscr{Q}$ that PLANEX($\mathscr{Q}$) reduces to PETRI in at most exponential time.

In view of Lemma 1, it suffices to reduce the problem of determining whether $(\mathfrak{o}, s_\omega)$ can be reached in Φ_Π given $\Pi \in \mathscr{Q}$ and a propositional state s_ω. This is equivalent to asking whether there exists a propositional state s_1 such that $(\mathfrak{o}, s_1, s_1)$ can be reached from $(\mathfrak{t}_0, s_0, s_\omega)$ in Φ_Π^{bi}. Again by Lemma 1 we can assume s_1 is given and reduce the problem of determining whether

$$(\mathfrak{o}, s_1, s_1) \text{ can be reached from } (\mathfrak{t}_0, s_0, s_\omega) \text{ in } \Phi_\Pi^{\text{bi}}. \tag{2}$$

Let $\Pi = (F,C,O,M,\delta,\mathfrak{t}_0,s_0)$ be an HTN problem. Let $\mathfrak{X}$ be the set of all nonempty task subnetworks of (initial or method) task networks of Π. For each $\mathfrak{x} \in \mathfrak{X}$, introduce a Petri place $p(\mathfrak{x})$. For each $s \subseteq F$ and $\upsilon \in \{0,1\}$, introduce a Petri place $p(s,\upsilon)$. Let P be the set of all places. A state $\sigma : P \to \mathbb{N}$ satisfying

$$\sum_{s \in \mathscr{P}(F)} \sigma\big(p(s,\upsilon)\big) = 1$$

for each $\upsilon < 2$, will encode an element of the bi-directional search space of Π. Namely, the task network is a disjoint union of $\sigma\big(p(\mathfrak{x})\big)$ copies of $\mathfrak{x}$ for each $\mathfrak{x} \in \mathfrak{X}$; the propositional state in forward search is the unique s such that $\sigma\big(p(s,0)\big) = 1$ and the propositional state in backward search is the unique s' such that $\sigma\big(p(s',1)\big) = 1$. Accordingly, let τ_0 be the unit state that encodes $(\mathfrak{t}_0, s_0, s_\omega)$ (naturally with $\tau_0\big(p(\mathfrak{t}_0)\big) = 1$), and τ_1 the unit state that encodes $(\mathfrak{o}, s_1, s_1)$.

Suppose that $s \subseteq F$ and $\mathsf{pr} \in O$ is applicable in s. Also let $\mathfrak{x} = (X, \prec, \alpha) \in \mathfrak{X}$ and $x \in X$ with name $\alpha(x) = \mathsf{pr}$. If $x \in X$ is $\prec$-minimal, we define the transition $\theta_0^{s,\mathfrak{x},x}$ by

$$\theta_0^{s,\mathfrak{x},x}\big(p(\mathfrak{y})\big) = \begin{cases} 1 & (\mathfrak{y} = \mathfrak{x} \setminus x) \\ -1 & (\mathfrak{y} = \mathfrak{x}) \\ 0 & (\text{otherwise}), \end{cases}$$

$$\theta_0^{s,\mathfrak{x},x}\big(p(s',0)\big) = \begin{cases} 1 & (s' = \gamma(s,\mathsf{pr})) \\ -1 & (s' = s) \\ 0 \quad (\text{otherwise}), & \end{cases}$$

$$\theta_0^{s,\mathfrak{x},x}\big(p(s',1)\big) = 0,$$

This transition corresponds to executing task x in forward search. If $x \in X$ is $\prec$-maximal, define the transition $\theta_1^{s,\mathfrak{x},x}$ by

$$\theta_1^{s,\mathfrak{x},x}\big(p(\mathfrak{y})\big) = \begin{cases} 1 & (\mathfrak{y} = \mathfrak{x} \setminus x) \\ -1 & (\mathfrak{y} = \mathfrak{x}) \\ 0 & (\text{otherwise}), \end{cases}$$

$$\theta_1^{s,\mathfrak{x},x}\big(p(s',0)\big) = 0,$$

$$\theta_1^{s,\mathfrak{x},x}\big(p(s',1)\big) = \begin{cases} 1 & (s' = s) \\ -1 & (s' = \gamma(s,\mathsf{pr})) \\ 0 \quad (\text{otherwise}). & \end{cases}$$

This transition corresponds to executing task x in backward search.

We shall see that firing transitions $\theta_\upsilon^{s,\mathfrak{x},x}$ gets rid of primitive tasks until a compound task is *isolated*. For each method $\mu = (\mathsf{c}, \mathfrak{t}) \in M$ and $\mathfrak{x} = (X, \prec, \alpha) \in \mathfrak{X}$ such that there exists an $\prec$-*isolated* $x \in X$ with name $\alpha(x) = \mathsf{c}$, introduce a transition $\theta_{\mathfrak{x},x,\mu}$ with

$$\theta_{\mathfrak{x},x,\mu}\big(p(\mathfrak{y})\big) = \begin{cases} 1 & (\mathfrak{y} = \mathfrak{t}) \\ 1 & (\mathfrak{y} = \mathfrak{x} \setminus x) \\ -1 & (\mathfrak{y} = \mathfrak{x}) \\ 0 & (\text{otherwise}), \end{cases}$$

$$\theta_{\mathfrak{x},x,\mu}\big(p(s,\upsilon)\big) = 0.$$

(We have to include the subscript x because if $\mathscr{Q} \neq \mathscr{H}_1$ the initial task network $\mathfrak{t}_0 \in \mathfrak{X}$ may contain multiple tasks with name c.) Note that x is isolated in the encoded task network because x is assumed to be isolated in $\mathfrak{r}$ and there is no order between the various $\mathfrak{r}$. Hence it should be clear that firing $\theta_{\mathfrak{r},x,\mu}$ corresponds to an application of the method μ to x in one of the copies of $\mathfrak{r}$ in the encoded task network.

Let Θ be the set of all transitions introduced above and $\mathscr{N} = (P, \Theta)$.

We claim that (2) holds iff τ_1 can be reached from τ_0 in the search space $\Phi_{\mathscr{N}}$.

Since the firings of $\mathscr{N}$ translate into movements of Φ^{bi}_{Π}, the direction "$\Leftarrow$" is clear. This direction of the proof actually works for any HTN problem Π.

Conversely, assume (2). First consider the case $\Pi \in \mathscr{H}_1$. Then any task network reachable from $(\mathfrak{t}_0, s_0, s_\omega)$ in Φ^{bi}_{Π} has at most one compound task. Decomposing a compound task x with a method can always be deferred until x is isolated: if there is a (primitive) predecessor task $t \prec x$, then t can be executed in forward search before decomposing x, since all primitive tasks executed before t have to be already present in the task network because there is no compound task besides x in the task network; similarly, if there is a (primitive) successor task, it can be executed in backward search before decomposing x. Such a solution is exactly what $\mathscr{N}$ captures.

Next suppose that $\Pi \in \mathscr{I}$. Then solving Π with backward search and deferring decompositions for as long as possible implies that again only isolated compound tasks will be decomposed. Thus $\mathscr{N}$ encodes the solution. The argument for $\mathscr{F}$ is analogous using forward search. □

Hardness

Next, we turn from showing membership (and thus decidability) of $\mathscr{I}, \mathscr{H}_1$, and $\mathscr{F}$ to showing hardness. While from Example 3, one might suspect that these problems could be computationally easy, we show that even clean, bottomless, one-hole-digging, loop-unrolling problems are in general much more complex. To be precise, we show that even this highly restricted class of problems is also hard for the class $\mathbb{ACKERMANN}$.

To establish this result, we show that the reachability problem for Petri nets can be encoded in such an HTN planning problem. For transparency, we don't reduce from the general PETRIGEN, but restrict ourselves to ordinary Petri nets with unit states, i.e. PETRI.

Note that parts of the construction – notably the concept of trashing – are only necessary as we want to establish hardness even for loop-unrolling problems. This proof translates to an easier version without the loop-unrolling property and without trashing.

In Figure 1 we provide an example for a plan that simulates a concrete Petri net, which might be helpful to the reader. An explanation of the example can be found at the end of the proof. Further Figure 2 shows the tasks contained in the construction's only recursive method task network while Table 2 shows the preconditions and effects of all actions compactly.

Proposition 5. *PETRI reduces to PLANEX($\mathscr{C} \cap \mathscr{H}_1 \cap \mathscr{L} \cap \mathscr{B}$) in polynomial time.*

Proof. Let $\mathscr{N} = (P, \Theta)$ be an ordinary Petri net and τ_0 and τ_1 unit states of $\mathscr{N}$. We define $\Pi = (F, C, O, M, \delta, \mathfrak{t}_0, s_0) \in \mathscr{C} \cap \mathscr{H}_1 \cap \mathscr{L} \cap \mathscr{B}$. Let $C = \{\textbf{comp}\}$ and M as in (1). τ_1 will be reachable from τ_0 in $\Phi_{\mathscr{N}}$ iff a large number of applications of cont yields a solution to Π.

Let

$$F = \{\texttt{searchMode}, \texttt{pTrashMode}, \texttt{rTrashMode}, \quad (3)$$
$$\texttt{transInProg}, \texttt{pTrashInProg},$$
$$\texttt{inc}(p), \texttt{dec}(p) : p \in P\}.$$

We shall design our problem in such a way that any solution to Π can be split into three phases, that are characterized by the truth of the variables in (3) and separated by the tasks in O_{main}. searchMode simulates the search in $\Phi_{\mathscr{N}}$, pTrashMode trashes unused place tokens and rTrashMode trashes any remaining tasks.

$$O_{\text{main}} = \{\mathsf{startPTrashMode}, \mathsf{startRTrashMode}\},$$
$$O' = \{\mathsf{inc}(p), \mathsf{dec}(p), \mathsf{startPTrash}, \mathsf{endPTrash},$$
$$\mathsf{startTrans}, \mathsf{endTrans},$$
$$\mathsf{requestInc}(p), \mathsf{checkInc}(p),$$
$$\mathsf{requestDec}(p), \mathsf{checkDec}(p),$$
$$\mathsf{fakeInc}(p), \mathsf{fakeDec}(p) : p \in P\},$$
$$O = O_{\text{main}} \sqcup O'.$$

The cont network $\mathfrak{t}$ is given by Figure 2 where α removes subscripts. For any $p \in P$, the tasks $\mathsf{inc}(p)$ (to be read "increment p") and $\mathsf{dec}(p)$ (to be read "decrement p") of $\mathfrak{t}$ together encode a single occurrence of a token at place p (at least during searchMode). Specifically, for each $p \in P$, the value of p in a state of $\mathscr{N}$ is given by the number of copies of $\mathfrak{t}$ of which task $\mathsf{inc}(p)$ has been performed but task $\mathsf{dec}(p)$ has not. δ is given by Table 2 and $\mathfrak{t}_0 = (T_0, \prec_0, \alpha_0)$ with

$$T_0 = \{\textbf{comp}\} \sqcup$$
$$\Big(\{\mathsf{requestInc}(p) \prec_0 \mathsf{checkInc}(p) : p \in P \,\&\, \tau_0(p) = 1\} \prec_0 \quad (4)$$
$$\{\mathsf{requestDec}(p) \prec_0 \mathsf{checkDec}(p) : p \in P \,\&\, \tau_1(p) = 1\} \prec_0 \quad (5)$$
$$O_{\text{main}}\Big),$$

$\alpha_0 = \text{id}$ and $s_0 = \{\texttt{searchMode}\}$.

Observe that in searchMode we can only execute "token tasks" ($\mathsf{inc}(p)$, $\mathsf{dec}(p)$), "transition tasks" (right-most columns in Figure 2) and "boundary tasks" ((4)–(5)). The boundary tasks (4) generate τ_0 and the boundary tasks (5) consume τ_1. It is w.l.o.g. that a plan starts with (4) and the searchMode ends with (5) (both accompanied by executions of token tasks). Firing a transition θ corresponds to executing an entire chain of six transition tasks of θ, also accompanied by executions of token tasks. The variable transInProg prevents that we work on two transitions at the same time. I.e. once $\mathsf{startTrans}_\theta$ is executed, the task $\mathsf{endTrans}_\theta$ in the same copy of $\mathfrak{t}$ has to be executed before any other $\mathsf{startTrans}_{\theta'}$ can be executed. The intermediate transition tasks ensure that the Petri net state encoded by the HTN state is updated appropriately. All boundary tasks have to be executed before we execute

$\mathsf{cont}, \mathsf{stop},$
$\mathsf{startTrans}_\theta, \mathsf{requestInc}(p)_\theta, \mathsf{inc}(p), \mathsf{checkInc}(p)_\theta, \mathsf{endTrans}_\theta,$
$\underline{\mathsf{requestDec}(p)}, \mathsf{dec}(p), \underline{\mathsf{checkDec}(p)},$
$\underline{\mathsf{startPTrashMode}},$
$\mathsf{startPTrash}_q, \mathsf{requestInc}(q), \mathsf{inc}(q), \mathsf{checkInc}(q),$
$\mathsf{requestDec}(q), \mathsf{dec}(q), \mathsf{checkDec}(q), \mathsf{endPTrash}_q,$
$\underline{\mathsf{startRTrashMode}},$
$\mathsf{startPTrash}_p, \mathsf{requestInc}(p), \mathsf{fakeInc}(p), \mathsf{checkInc}(p),$
$\mathsf{requestDec}(p), \mathsf{fakeDec}(p), \mathsf{checkDec}(p), \mathsf{endPTrash}_p,$
$\mathsf{fakeInc}(q), \mathsf{fakeDec}(q),$
$\mathsf{startTrans}_\eta, \mathsf{requestInc}(q)_\eta, \mathsf{fakeInc}(q)_\eta, \mathsf{checkInc}(q)_\eta, \mathsf{endTrans}_\eta,$
$\mathsf{fakeInc}(p)_\theta$

Figure 1: Example solution to Π ($\underline{\text{tasks in } T_0}$).

	p	q
θ	1	0
η	0	1
τ_0	0	0
τ_1	1	0

Table 1: Example instance of PETRI.

$$\begin{tikzcd}
\boxed{\mathbf{comp}} & \boxed{\mathsf{inc}(p)} \arrow[d] & \boxed{\mathsf{startPTrash}_p} \arrow[d] & & \boxed{\mathsf{startTrans}_\theta} \arrow[d] & \\
& \boxed{\mathsf{dec}(p)} & \boxed{\mathsf{requestInc}(p)} \arrow[d] & \boxed{\mathsf{fakeInc}(p)} & \boxed{\mathsf{requestDec}(p_-)_\theta} \arrow[d] & \boxed{\mathsf{fakeDec}(p_-)_\theta} \\
& & \boxed{\mathsf{checkInc}(p)} \arrow[d] & & \boxed{\mathsf{checkDec}(p_-)_\theta} \arrow[d] & \\
& & \boxed{\mathsf{requestDec}(p)} \arrow[d] & \boxed{\mathsf{fakeDec}(p)} & \boxed{\mathsf{requestInc}(p_+)_\theta} \arrow[d] & \boxed{\mathsf{fakeInc}(p_+)_\theta} \\
& & \boxed{\mathsf{checkDec}(p)} \arrow[d] & & \boxed{\mathsf{checkInc}(p_+)_\theta} \arrow[d] & \\
& & \boxed{\mathsf{endPTrash}_p} & & \boxed{\mathsf{endTrans}_\theta} &
\end{tikzcd}$$

Figure 2: Task network $\mathrm{t} = (T, \prec, \alpha)$ (include instances for all $p, p_-, p_+ \in P$, $\theta \in \Theta$ with $\theta(p_+) = 1$, $\theta(p_-) = -1$).

pr	π_+	π_-	e_+	e_-
startPTrashMode	searchMode	transInProg	pTrashMode	searchMode
startRTrashMode	pTrashMode	pTrashInProg	rTrashMode	pTrashMode
startPTrash		searchMode pTrashInProg	pTrashInProg	
endPTrash				pTrashInProg
startTrans		pTrashMode transInProg	transInProg	
endTrans				transInProg
inc(p)	inc(p)	rTrashMode		inc(p)
dec(p)	dec(p)	rTrashMode		dec(p)
requestInc(p)		inc(p)	inc(p)	
checkInc(p)		inc(p)		
requestDec(p)		dec(p)	dec(p)	
checkDec(p)		dec(p)		
fakeInc(p)	rTrashMode			inc(p)
fakeDec(p)	rTrashMode			dec(p)

Table 2: Action mapping $\delta(\mathsf{pr}) = (\pi_+, \pi_-, e_+, e_-)$.

startPTrashMode. Hence the crucial claim is that an HTN state without remaining boundary tasks and with propositional state $\{$`searchMode`$\}$ encodes the zero state of $\mathscr{N}$ iff executing startPTrashMode allows one to reach o in Φ_Π. But in `pTrashMode`, only token tasks and tasks in the third column of Figure 2 can be executed. Hence in view of `pTrashInProg`, for each $p \in P$, equally many copies of inc(p) as dec(p) are executed in this phase, so (since they cannot be executed in the final phase) the HTN state must encode the zero state of $\mathscr{N}$ when entering `pTrashMode`. Conversely, it is easy to show that after trashing the tokens all remaining tasks can be finished in `rTrashMode` using the "fake" tasks.

As an example, suppose that $P = \{p, q\}$ and $\Theta = \{\theta, \eta\}$ and τ_0, τ_1 are as in Table 1. Then τ_1 is reachable from τ_0 by firing just θ. Hence to solve Π only one copy of t is needed. See Figure 1. Notice that at the start of `pTrashMode`, both slot tasks for p have been executed while neither slot task for q has been executed; at the start of `rTrashMode`, all slot tasks have been executed. □

The method task network t in Figure 2 has the shape of *parallel sequences*. This type of structure occurs frequently in HTN planning; cf. (Behnke et al. 2022).

Conclusion and Future Work

We introduced several new fragments of HTN planning and proved that multiple combinations of them are complete for the large class $\mathbb{ACKERMANN}$ of decidable problems:

Theorem 6. *Let $\mathscr{C} \cap \mathscr{H}_1 \cap \mathscr{L} \cap \mathscr{B} \subseteq \mathscr{Q} \subseteq \mathscr{H}_1 \cup \mathscr{I} \cup \mathscr{F}$. Then PLANEX$(\mathscr{Q})$ is $\mathbb{ACKERMANN}$-complete.*

Proof. Theorem 2 and Propositions 4 and 5. □

Table 3 lists some natural fragments of HTN planning. Notice that the new fragments have higher computational complexity than the previously known ones. Our proof relies on the recently established $\mathbb{ACKERMANN}$-completeness of the Petri net reachability problem. In future work we hope to further investigate the relationship between HTN planning and Petri nets, and use it to invent new algorithms for HTN planning. In particular, the construction in the proof of Proposition 4 can be carried out for any HTN problem, and may provide a heuristic.

Moreover, we would like to investigate the class $\mathscr{H}_2$ of *two-hole-digging* problems, that have two compound tasks in the initial task network but only one compound task per method task network. It is known that PLANEX$(\mathscr{H}_2)$ is undecidable; see (Höller et al. 2023). However, is PLANEX$(\mathscr{H}_2 \cap \mathscr{L})$ decidable?

References

Alford, R. 2013. *Search complexities for HTN planning*. Ph.D. thesis.

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo, S.; and Aha, D. W. 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 2016)*, 20–28. AAAI Press.

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

Alford, R.; Kuter, U.; and Nau, D. 2009. Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1629–1634. AAAI Press.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2012. HTN Problem Spaces: Structure, Algorithms, Termination. In *Proceedings of the 5th Annual Symposium on Combinatorial Search (SoCS 2012)*, 2–9. AAAI Press.

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

Behnke, G.; Pollitt, F.; Höller, D.; Bercher, P.; and Alford, R. 2022. Making Translations to Classical Planning Competitive With Other HTN Planners. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, 9687–9697. AAAI Press.

Czerwiński, W.; and Orlikowski, L. 2022. Reachability in Vector Addition Systems is Ackermann-complete. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, 1229–1240.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN Planning: Complexity and Expressivity. In *Proceedings of the Association for the Advancement of Artificial Intelligence*.

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

Esparza, J. 1998. Decidability and Complexity of Petri Net Problems – an Introduction. *Lecture Notes in Computer Science*, 1491: 374–428.

Gazen, B.; and Knoblock, C. 1997. Combining the expressivity of UCPOP with the efficiency of graphplan. In *Proceedings of the European Conference on Planning: Recent Advances in AI Planning*, volume 4, 221–233. Springer.

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 2011)*, 1955–1961. AAAI Press.

Höller, D. 2021. Translating Totally Ordered HTN Planning Problems to Classical Planning Problems Using Regular Approximation of Context-Free Languages. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 159–167. AAAI Press.

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 2014)*, volume 263, 447–452. IOS Press.

Fragment $\mathscr{Q}$	Complexity of PLANEX($\mathscr{Q}$)	Reference
All HTN problems	Undecidable	(Erol, Hendler, and Nau 1994)
$\mathscr{H}_1$ (one-hole-digging)	$\mathbb{ACKERMANN}$	Theorem 6
$\mathscr{I}$ (initial)	$\mathbb{ACKERMANN}$	Theorem 6
$\mathscr{F}$ (final)	$\mathbb{ACKERMANN}$	Theorem 6
$\mathscr{C}$ (clean)	$\mathbb{ACKERMANN}$	Theorem 6
Tail-recursive	$\mathbb{EXPSPACE}$	(Alford, Bercher, and Aha 2015)
Acyclic	$\mathbb{NEXP}$	(Alford, Bercher, and Aha 2015)
Total order	$\mathbb{EXP}$	(Alford, Bercher, and Aha 2015)
Regular	$\mathbb{PSPACE}$	(Erol, Hendler, and Nau 1994)

Table 3: Some fragments of HTN planning and their complexities.

Höller, D.; Lin, S.; Erol, K.; and Bercher, P. 2023. From PCP to HTN Planning Through CFGs. *The 10th International Planning Competition – Planner and Domains Abstracts*.

Leroux, J.; and Schmitz, S. 2019. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*, 50, 1–13.

Magnaguagno, M. C.; Meneguzzi, F.; and de Silva, L. 2021. HyperTensioN: A three-stage compiler for planning. In *Proceedings of 10th International Planning Competition: planner and domain abstracts (IPC 2020)*.

Murata, T. 1989. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77, 541–580.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999)*, 968–973.

Pohl, I. 1969. Bi-directional and Heuristic Search in Path Problems.

Sacerdoti, E. D. 1975. *A structure for plans and behavior*. Ph.D. thesis, Department of Computer Science, Stanford University.

Schmitz, S. 2016. Complexity Hierarchies beyond Elementary. *ACM Transactions on Computation Theory*, 8(1).

Schreiber, D. 2021. Lifted Logic for Task Networks: TOHTN Planner Lilotane in the IPC 2020. In *Proceedings of 10th International Planning Competition: planner and domain abstracts (IPC 2020)*.