Barely Decidable Fragments of Planning

2025 erratum corrected version: sentence "Interestingly all [...] in PSPACE." retracted from section 5

Maurice Dekker[a,*] **and Gregor Behnke**[a]

[a]University of Amsterdam
ORCID (Maurice Dekker): https://orcid.org/0009-0009-4050-7635, ORCID (Gregor Behnke): https://orcid.org/0000-0002-1445-9934

Abstract. Both numeric planning and Hierarchical Task Network (HTN) planning are highly expressive planning formalisms – at the cost of being undecidable in general. For both formalisms, decidable fragments are known. Studying these restricted fragments has lead to valuable insights, which ultimately gave rise to the development of new efficient planning algorithms.

We identify new decidable fragments of both numeric and HTN planning. For HTN planning, we introduce the fragments of *one-hole-digging*, *initial*, and *final* problems. The former restrict every task network to have at most one compound task, while the latter two restrict compound tasks to be order-minimal or order-maximal, respectively. For numeric planning, we introduce Positive Numeric Planning (PNP) where the value of numeric variables can only be non-negative. We determine the complexity of these fragments: they are Ackermann-complete – which is significantly more difficult than any prior known decidable fragment, but still barely decidable.

1 Introduction

While the expressive power of classical planning is limited [9], there are several highly expressive extensions to it. In this paper, we will study two such extensions: Hierarchical Task Network (HTN) planning [34, 11] and numeric planning [21, 18]. HTN planning 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. Numeric planning allows for using integer-valued variables to describe states and for actions that manipulate them. While we will concentrate mainly on HTN planning, our results show an interesting connection between the two, previously unconnected, extensions of classical planning – via the reachability problem of Petri nets [17, 13]. While classical planning relates to safe Petri nets [20], we will work with general Petri nets.

The plan existence problems for both HTN and numeric planning are in general undecidable. In several cases, theoretical insights into decidable fragments have informed practical planning methods and have lead to new algorithms. This includes Erol's insight that totally-ordered HTN planning is decidable [12], which has lead to dedicated total-order HTN planners [30, 27, 37, 6, 5, 2], but also the complexity analysis of tail-recursive HTN problems [4] and the idea of relating HTN planning to formal languages [23], which have lead to dedicated planning algorithms ([5] and [22] respectively).

* Corresponding Author. Email: p.m.dekker@uva.nl.

For both HTN planning and numeric planning, we present new, previously unknown, classes of decidable problems. For HTN planning these are *one-hole-digging*, *initial*, and *final* problems, while for numeric planning we present Positive Numeric Planning (PNP). These classes are orthogonal to the previously investigated classes and thus illuminate a new island of decidability. While being decidable, one-hole-digging, initial, and final HTN problems and PNP are only barely decidable: we show that they are complete for ACKERMANN, i.e., for all problems solvable by a program whose runtime is limited by an Ackermann function. For PNP, we are also able to identify a computationally easier subclass (PNP without equals constraints in the goal) that is "only" EXPSPACE-complete.

We first cover our results for HTN planning, and then turn to numeric planning in Sec. 5.

2 Preliminaries

In order to present the new fragments and results for HTN planning, 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.

2.1 HTN planning

In this section, we set up the HTN formalism following Geier and Bercher [16]. 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 (i) T is a finite set of *tasks*; (ii) $\prec$ is a strict partial order on T; and (iii) $\alpha : T \to N$ assigns a task name to each task in the network. Let TN_N be the set of all task networks over N. If $T' \subseteq T$, we define the *task subnetwork*

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

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

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

and $\phi_i(t_i) \not\prec \phi_j(t_j)$ whenever $i, j \in I$ are distinct and t_i and t_j are tasks of $\mathfrak{t}_i$ and $\mathfrak{t}_j$ respectively.

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

- F, C, and O are finite sets of *(propositional) variables*, *compound task names*, and *primitive task names*, respectively;
- $M \subseteq C \times \mathrm{TN}_{C \sqcup O}$ is a finite set of *(decomposition) methods*;
- $\delta : O \to \mathscr{P}(F)^4$ is an *action mapping*;
- $\mathfrak{t}_0 \in \mathrm{TN}_{C \sqcup O}$ 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: propositional variables are typewriter blue, compound task names bold and **brown**, primitive task names sans serif pink and decomposition methods green. Let $\mathfrak{t} = (T, \prec, \alpha)$ be a task network over $C \sqcup O$. It is called a task network *of* Π if $\mathfrak{t} = \mathfrak{t}_0$ 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).

A decomposition method is applied to change 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 exist embeddings $\phi_1 : \mathfrak{t}_1 \setminus t \hookrightarrow \mathfrak{t}_2$ and $\phi : \mathfrak{t} \hookrightarrow \mathfrak{t}_2$ such that

$$T_2 = \operatorname{Im}\phi_1 \sqcup \operatorname{Im}\phi$$

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

$$t_1 *_1 t \Leftrightarrow \phi_1(t_1) *_2 \phi(t').$$

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

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

The action mapping δ associates with each pr $\in O$ a four-tuple $\delta(\mathsf{pr}) = (\pi_+, \pi_-, e_+, e_-)$ consisting of a *positive precondition* π_+, a *negative precondition* π_-[1], a *positive effect* e_+, and a *negative effect* e_-. Now if $s \subseteq F$ satisfies $\pi_+ \subseteq s$ and $\pi_- \cap s = \emptyset$, we say pr is *applicable* in s and define $\gamma(s, \mathsf{pr}) = (s \setminus e_-) \cup e_+$. 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

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

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

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

The *search graph* of Π is $\Phi_\Pi = \big(\Omega_\Pi, \rightsquigarrow_\Pi\big)$. For the relation with other views on HTN search, we refer to [3]. The problem Π is *solvable* if for some propositional state s_ω, the state $(\mathfrak{o}, s_\omega)$ is reachable from $(\mathfrak{t}_0, s_0)$ 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.

Example 1. *Let* $F = \{\mathtt{var}\}$, $C = \{\mathbf{comp}\}$, $O = \{\mathsf{pr}, \mathsf{g}\}$,

$$M = \big\{\mathit{stop} = (\mathbf{comp}, \mathfrak{o}), \mathit{cont} = (\mathbf{comp}, \mathfrak{t})\big\} \tag{1}$$

where $\mathfrak{t}$ *contains two ordered tasks with respective names* **comp** *and* pr, $\delta(\mathsf{pr}) = \langle \emptyset, \emptyset, F, \emptyset \rangle$, $\delta(\mathsf{g}) = \langle F, \emptyset, \emptyset, \emptyset \rangle$, $\mathfrak{t}_0$ *contains three unordered tasks named* **comp**, **comp** *and* g, *and* $s_0 = \emptyset$. *Then* $\Pi = (F, C, O, M, \delta, \mathfrak{t}_0, s_0)$ *is an HTN problem. To solve this problem, one should use the method* cont *to introduce a task with name* pr. *Executing this task makes the precondition* var *of* g *true. The method* stop *can be used to get rid of remaining compound tasks. In fact, every path through* Φ_Π *starting at* $(\mathfrak{t}_0, s_0)$ *leads to the goal, except when one immediately applies the* stop *method twice or when one applies the* cont *method infinitely often.*

If $\mathscr{Q}$ is a class of planning 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 for various classes $\mathscr{Q}$ of HTN problems, whose complexities range from polynomial time to undecidable [1, 4, 11, 16].

2.2 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 $\big|\theta(p)\big| \leq 1$ for all $\theta \in \Theta$ and $p \in P$. Ordinary Petri nets have the same modelling power as arbitrary Petri nets (see [29, Sect. 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 a Petri net $\mathscr{N}$ and states τ_0, τ_1 of $\mathscr{N}$ whether or not τ_1 can be reached from τ_0 in $\Phi_\mathscr{N}$. PETRI_1 is the same problem under the additional assumptions that $\mathscr{N}$ is ordinary and τ_0 and τ_1 are unit states. For more information on this and similar problems we refer to [13]. It is easy to see that PETRI reduces to PETRI_1 in polynomial time. Moreover, the following is standard:

Lemma 2. *Given a finite set* P *of instances of* PETRI_1 *we can compute in polynomial time another instance of* PETRI_1 *that has answer "yes" iff some instance in* P *has answer "yes".*

2.3 Complexity theory

In this paper, we will consider complexity classes that lie beyond the typically considered hierarchy of complexity classes such as L, NL, P, NP, PSPACE, EXPTIME, ... We introduce the notion of ACKERMANN-completeness following [36].

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

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

As a result, we get that $F_1(n) \geq 2n$, $F_2(n) \geq 2^n$ and

$$F_3(n) \geq \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n \text{ times}},$$

while also $\lim_{k\to\infty} F_k(0) = \infty$. Lastly we define $F_\omega(n) = F_n(n)$, which is the Ackermann function. Let 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

[1] It is well-known that negative preconditions can be compiled away in linear time; cf. [15, section 2.6].

k. 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 runtime limited to some fixed height exponentiation tower. Problems in ACKERMANN can be still much harder.

A problem Π is ACKERMANN-hard if for every problem $\Pi' \in$ 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 ACKERMANN-hardness.

Leroux and Schmitz [25] proved that PETRI$_1 \in$ ACKERMANN. Recently, Czerwiński and Orlikowski complemented this result with hardness:

Theorem 3. *[10] PETRI is ACKERMANN-complete.*

Hence the same holds true for PETRI$_1$.

3 New fragments of HTN planning

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:

$$\forall \mathfrak{t} = (T, \prec, \alpha) \in \mathrm{TN}_{C \sqcup O} : \Big(\mathfrak{t}_0 = \mathfrak{t} \mid \big(\exists \mathbf{c} : (\mathbf{c}, \mathfrak{t}) \in M\big) \Big)$$
$$\implies \forall t, t' \in T : \big(t \prec t' \implies \alpha(t') \in O\big).$$

- 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. Equivalently, all compound tasks are isolated (a task is *isolated* in a task network if it is not ordered against any other task in the task network).
- Π is *one-hole-digging* if every task network of Π contains at most one compound task:

$$\big|\alpha_0^{-1}[C]\big| \leq 1 \;\&\; \forall \big(\mathbf{c}, (T, \prec, \alpha)\big) \in M : \big|\alpha^{-1}[C]\big| \leq 1.$$

- Π is *bottomless* if every primitive method task network is empty:

$$\forall (\mathbf{c}, \mathfrak{t}) \in M : \mathfrak{t} \in \mathrm{TN}_O \implies \mathfrak{t} = \mathfrak{o}.$$

- Π is *loop-unrolling* if it contains at most one compound task name and at most two methods:

$$|C| \leq 1 \;\&\; |M| \leq 2.$$

The problem Π from Ex. 1 is initial.

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 equation (1) holds for some comp and $\mathfrak{t}$. Let $\mathfrak{t}^{\mathrm{pr}}_{(0)}$ be the largest primitive task subnetwork of $\mathfrak{t}_{(0)}$; if Π is additionally clean, 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 4. *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, so the propositional variables are not able to capture the amount of dirt that is dug. The propositional state will be a highly simplified model of the world. However, the task hierarchy can make sure that as many dirt is dug up as is put back. Let* $F = \{\mathtt{hole}, \mathtt{buried}\}$, $C = \{\mathit{bury}\}$, $O = \{\mathit{dig}, \mathit{put}, \mathit{cover}\}$,

$$M = \big\{\mathit{deeper} = (\mathit{bury}, \mathfrak{t}_{\mathit{deeper}}), \mathit{bottom} = (\mathit{bury}, \mathfrak{t}_{\mathit{bottom}})\big\}$$

where $\mathfrak{t}_{\mathit{deeper}}$ *is the totally ordered task network consisting of three tasks with names* dig, bury *and* cover, $\mathfrak{t}_{\mathit{bottom}}$ *is the task network consisting of one task with name* put, $\delta(\mathit{dig}) = \langle \emptyset, \emptyset, \{\mathtt{hole}\}, \emptyset \rangle$, $\delta(\mathit{put}) = \langle \{\mathtt{hole}\}, \emptyset, \{\mathtt{buried}\}, \emptyset \rangle$, $\delta(\mathit{cover}) = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$, $\mathfrak{t}_0$ *is the task network consisting of one task* t_0 *with name* $\alpha_0(t_0) = \mathit{bury}$ *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 infinitely often.*

Remark 5. *Every classical STRIPS planning problem can be compiled into an equivalent clean problem, an equivalent totally ordered initial problem, and an equivalent totally ordered final problem. This is done in essentially the same way as the original translation of STRIPS to HTN [12, Thm. 5].*

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 [11], 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 [11]). 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$(\mathscr{F}') \in$ 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 EXPTIME [12], so much easier than ACKERMANN.

4 Results for HTN planning

Intuitively, the class $\mathscr{H}_1$ might be complexity-wise close to other HTN classes or even classical (i.e. non-hierarchical) 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. Ex. 4 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 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 Thm. 3, by showing equivalence of PLANEX($\mathscr{H}_1$) to PETRI$_1$. As a bonus, our reductions work for $\mathscr{I}$ and $\mathscr{F}$ as well.

4.1 Membership

In this section we prove membership in ACKERMANN (and hence decidability) of the plan existence problem of a large fragment of HTN planning. The proof combines progression and regression. Bi-directional search is a common technique in planning known since 1969 [31]. If $\Pi = (F, C, O, M, \delta, \mathfrak{t}_0, s_0)$ is an HTN problem, its *bi-directional search space* is

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

The first propositional state in a triple in Ω^{bi}_Π is understood as the propositional state in forward search and the second in backward search. The search graph $\Phi^{\text{bi}}_\Pi = (\Omega^{\text{bi}}_\Pi, \rightsquigarrow^{\text{bi}}_\Pi)$ 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:

$$\Big(\mathfrak{t}, s, \gamma(s_1, \alpha(t))\Big) \rightsquigarrow^{\text{bi}}_\Pi (\mathfrak{t} \smallsetminus t, s, s_1)$$

whenever $\mathfrak{t} = (T, \prec, \alpha)$ and $t \in T$ is $\prec$-maximal. It is easy to see that Π is solvable iff $(\mathfrak{o}, s_1, s_1)$ can be reached from $(\mathfrak{t}_0, s_0, s_\omega)$ in Φ^{bi}_Π for some propositional states s_1, s_ω. In particular, the bi-directional search starts with the propositional states s_0 and s_ω of the start and the end of the plan, the latter of which has to be guessed.

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^{bi}_Π.

In general, such a bi-directional HTN search cannot be encoded by a Petri net, since decompositions allow the task network to become arbitrarily complicated. However, we will show that a Petri net can capture a method application to a compound task that is *isolated* in the task network of the bi-directional search state. The Petri net does this by spawning a token to keep track of progression and regression within the method task network. It turns out that these more restrictive method applications suffice for the HTN fragments we consider:

Lemma 6. *Every solvable $\Pi \in \mathscr{H}_1 \cup \mathscr{I} \cup \mathscr{F}$ can be solved in the bi-directional search graph of Π without method applications*

$$(\mathfrak{t}_1, s, s_1) \rightsquigarrow^{bi}_\Pi (\mathfrak{t}_2, s, s_1)$$

unless $\mathfrak{t}_1 \to_{t/\mu} \mathfrak{t}_2$ for some method μ of Π and some task t that is isolated in $\mathfrak{t}_1$.

Proof. First consider the case $\Pi \in \mathscr{H}_1$, and suppose that s_ω is a propositional state such the goal can be reached from $(\mathfrak{t}_0, s_0, s_\omega)$ in Φ^{bi}_Π. 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 can always be deferred until x is isolated: if there is a (primitive) predecessor task $t \prec x$, then t can be progressed before decomposing x, since all primitive tasks executed before t in the plan 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.

Next suppose that $\Pi \in \mathscr{F}$. Solve Π using only progression. Decomposing a compound task x can always be deferred until x is order-minimal (and hence isolated): if there are no isolated compound tasks in the task network, there are no minimal compound tasks so the next primitive task in the plan must already be present in the task network.

The argument for $\mathscr{I}$ is analogous using regression. □

We are now ready to show how a path through the search graph of a suitable Petri net provides a path through the bi-directional search graph of a one-hole-digging, initial or final HTN problem.

Proposition 7. *PLANEX($\mathscr{H}_1 \cup \mathscr{I} \cup \mathscr{F}$) reduces to PETRI$_1$ in exponential time.*

Proof. Write $\Pi = (F, C, O, M, \delta, \mathfrak{t}_0, s_0)$.

Consider the problem of determining given $\Pi \in \mathscr{H}_1 \cup \mathscr{I} \cup \mathscr{F}$ and propositional states $s_1, s_\omega \subseteq F$ whether

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

By Lemma 2, it suffices to reduce this problem to PETRI$_1$.

Let Π be any HTN problem. Let $\mathfrak{X}$ be the set of all nonempty task subnetworks of (initial or method) task networks of Π. Define a set of places as

$$P = \mathfrak{X} \sqcup \big(\mathscr{P}(F) \times \{\text{forward}, \text{backward}\}\big).$$

A state $\sigma : P \to \mathbb{N}$ satisfying

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

for each $\upsilon \in \{\text{forward}, \text{backward}\}$, will encode an element of the bi-directional search space of Π. Namely, the task network is a disjoint union of $\sigma(\mathfrak{x})$ copies of $\mathfrak{x}$ for each $\mathfrak{x} \in \mathfrak{X}$; the propositional state in forward search is the unique s such that $\sigma(s, \text{forward}) = 1$ and the propositional state in backward search is the unique s' such that $\sigma(s', \text{backward}) = 1$. Accordingly, let τ_0 be the unit state given by

$$\tau_0(p) = \begin{cases} 1 & (p = \mathfrak{t}_0) \\ 1 & \big(p = (s_0, \text{forward})\big) \\ 1 & \big(p = (s_\omega, \text{backward})\big) \\ 0 & (\text{otherwise}), \end{cases}$$

encoding $(\mathfrak{t}_0, s_0, s_\omega)$; and let τ_1 be the unit state given by

$$\tau_1(p) = \begin{cases} 1 & \big(p \in \{s_1\} \times \{\text{forward}, \text{backward}\}\big) \\ 0 & (\text{otherwise}), \end{cases}$$

encoding $(\mathfrak{o}, s_1, s_1)$.

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 a transition $P \to \mathbb{Z}$ by

$$p \mapsto \begin{cases} -1 & (p = \mathfrak{x}) \\ 1 & (p = \mathfrak{x} \smallsetminus x) \\ -1 & \big(p = (s, \text{forward})\big) \\ 1 & \Big(p = \big(\gamma(s, \mathsf{pr}), \text{forward}\big)\Big) \\ 0 & (\text{otherwise}), \end{cases}$$

Firing this transition corresponds to progressing task x of one of the copies of $\mathfrak{x}$ in the encoded task network. If $x \in X$ is $\prec$-maximal, define

a transition $P \to \mathbb{Z}$ by

$$p \mapsto \begin{cases} -1 & (p = \mathfrak{x}) \\ 1 & (p = \mathfrak{x} \setminus x) \\ -1 & \Big(p = \big(\gamma(s, \mathsf{pr}), \text{backward}\big)\Big) \\ 1 & \big(p = (s, \text{backward})\big) \\ 0 & (\text{otherwise}). \end{cases}$$

Firing this transition corresponds to regressing task x of one of the copies of $\mathfrak{x}$ in the encoded task network.

By Lemma 6, for $\Pi \in \mathscr{H}_1 \cup \mathscr{I} \cup \mathscr{F}$ it suffices to define transitions modelling decompositions of isolated tasks. Consider a method $\mu = (\mathfrak{c}, \mathfrak{t}) \in M$, a task network $\mathfrak{x} = (X, \prec, \alpha) \in \mathfrak{X}$ and a task $x \in X$ with name $\alpha(x) = \mathfrak{c}$. If $\sigma : P \to \mathbb{N}$ is a state encoding an element of Ω^{bi}_{Π}, then any of the $\sigma(\mathfrak{x})$ copies of x is isolated iff x is $\prec$-isolated. If this is the case, introduce a transition $P \to \mathbb{Z}$:

$$p \mapsto \begin{cases} -1 & (p = \mathfrak{x}) \\ 1 & (p = \mathfrak{t}) \\ 1 & (p = \mathfrak{x} \setminus x) \\ 0 & (\text{otherwise}). \end{cases}$$

Firing this transition corresponds to an application of the method μ to x in one of the copies of $\mathfrak{x}$ in the encoded task network.

Let Θ be the set of all transitions introduced above and $\mathscr{N} = (P, \Theta)$. Then (2) holds iff τ_1 can be reached from τ_0 in the search graph $\Phi_{\mathscr{N}}$. □

4.2 Hardness

Next, we turn from showing ACKERMANN-membership of the plan existence problems of $\mathscr{I}, \mathscr{H}_1$, and $\mathscr{F}$, to showing hardness. While from Exs. 1 and 4, 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 rather complex. To be precise, we show that even this highly restricted class of problems is also hard for the class 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 restrict ourselves to ordinary Petri nets with unit states. I.e. we reduce PETRI$_1$, not PETRI.

Fig. 1 shows the tasks contained in the construction's only recursive method task network while Table 1 compactly displays the preconditions and effects of all actions. Certain ingredients – notably the concept of trashing – are only necessary as we want to establish hardness even for loop-unrolling problems. The proof translates to an easier version without the loop-unrolling property and without trashing.

Proposition 8. *PETRI$_1$ 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 $C = \{\mathsf{comp}\}$. M will be of the form (1), such that τ_1 will be reachable from τ_0 in $\Phi_{\mathscr{N}}$ iff a sufficiently large number of applications of cont yields a solution to Π.

We denote our variables and tasks as follows:

$$F = \big\{\texttt{searchPhase}, \texttt{pTrashPhase}, \texttt{rTrashPhase}, \quad (3)$$

(flags for Petri net search, place trashing, and remaining trashing)

$$\texttt{transInProg}\ (\text{“transition firing in progress”}),$$

$$\texttt{pTrashInProg}\ (\text{“place trashing in progress”}),$$

$$\texttt{inc}(p)\ (\text{“increment } p\text{”}), \texttt{dec}(p)\ (\text{“decrement } p\text{”}) : p \in P\big\},$$

$$O_{\text{main}} = \{\mathsf{startPTrashPhase}, \mathsf{startRTrashPhase}\},$$

$$O' = \big\{\mathsf{inc}(p), \mathsf{dec}(p), \mathsf{startPTrash}, \mathsf{endPTrash},$$

$$\mathsf{startTrans}, \mathsf{endTrans},$$

$$\mathsf{requestInc}(p), \mathsf{checkInc}(p),$$

$$\mathsf{requestDec}(p), \mathsf{checkDec}(p),$$

$$\mathsf{fakeInc}(p), \mathsf{fakeDec}(p) : p \in P\big\},$$

$$O = O_{\text{main}} \sqcup O'.$$

The cont network $\mathfrak{t}$ is given by Fig. 1 where α removes subscripts. δ is given by Table 1 and $\mathfrak{t}_0 = (T_0, \prec_0, \alpha_0)$ with

$$T_0 = \{\mathsf{comp}\} \sqcup$$

$$\Big(\big\{\mathsf{requestInc}(p) \prec_0 \mathsf{checkInc}(p) : p \in P \,\&\, \tau_0(p) = 1\big\} \prec_0 \quad (4)$$

$$\big\{\mathsf{requestDec}(p) \prec_0 \mathsf{checkDec}(p) : p \in P \,\&\, \tau_1(p) = 1\big\} \prec_0 \quad (5)$$

$$O_{\text{main}}\Big),$$

$\alpha_0 = \text{id}$ and $s_0 = \{\texttt{searchPhase}\}$. We claim that τ_1 is reachable from τ_0 in $\Phi_{\mathscr{N}}$ iff Π is solvable.

As a guiding example, suppose that $P = \{p, q\}$ and $\Theta = \{\theta, \eta\}$ and τ_0, τ_1 are as in Table 2. Then τ_1 is reachable from τ_0 by firing just θ. Fig. 2 shows a solution to Π. Only one copy of $\mathfrak{t}$ is needed in this case (see line 1).

For any $p \in P$, the "token tasks" $\mathsf{inc}(p)$ and $\mathsf{dec}(p)$ of $\mathfrak{t}$ together encode a single occurrence of a token at place p. Specifically, given the task network of a node in the search graph of Π reachable from $(\mathfrak{t}_0, s_0)$, we can define a state of $\mathscr{N}$ by setting the number of tokens at place $p \in P$ to the number of copies of $\mathfrak{t}$ of which task $\mathsf{inc}(p)$ has been performed but task $\mathsf{dec}(p)$ has not. The variables $\texttt{inc}(p)$ and $\texttt{dec}(p)$ can be thought of as commands for the incrementation respectively decrementation of place p. A pair of two tasks with names $\mathsf{requestInc}(p)$ and $\mathsf{checkInc}(p)$ can be executed in this order iff in the interim either the value of p in the encoded Petri net state is incremented or a task with name $\mathsf{fakeInc}(p)$ is executed. Here the importance of $\mathsf{checkInc}(p)$ is to leave the HTN no choice but to comply to the incrementation command. Analogously for decrementation. In Fig. 2 this happens in lines 2, 3, 5 and 6, which we will come back to.

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 executions of the tasks in O_{main} (lines 4 and 7 in Fig. 2). $\texttt{searchPhase}$ simulates the search in $\Phi_{\mathscr{N}}$ from τ_0 to τ_1. Then $\texttt{pTrashPhase}$ trashes unused token tasks. Finally, $\texttt{rTrashPhase}$ trashes any remaining tasks. We next detail each of these phases.

Observe that in $\texttt{searchPhase}$ we can only execute token tasks, "transition tasks" (fifth column in Fig. 1) and "boundary tasks" ((4)–(5)). The boundary tasks (4) generate τ_0 and the boundary tasks (5) consume τ_1 (line 3 in Fig. 2). The order $\prec_0$ ensures that the boundary tasks are completed in $\texttt{searchPhase}$. Hence, since tasks $\mathsf{inc}(p)$ are $\prec$-minimal and tasks $\mathsf{dec}(p)$ are $\prec$-maximal, it is w.l.o.g. that a

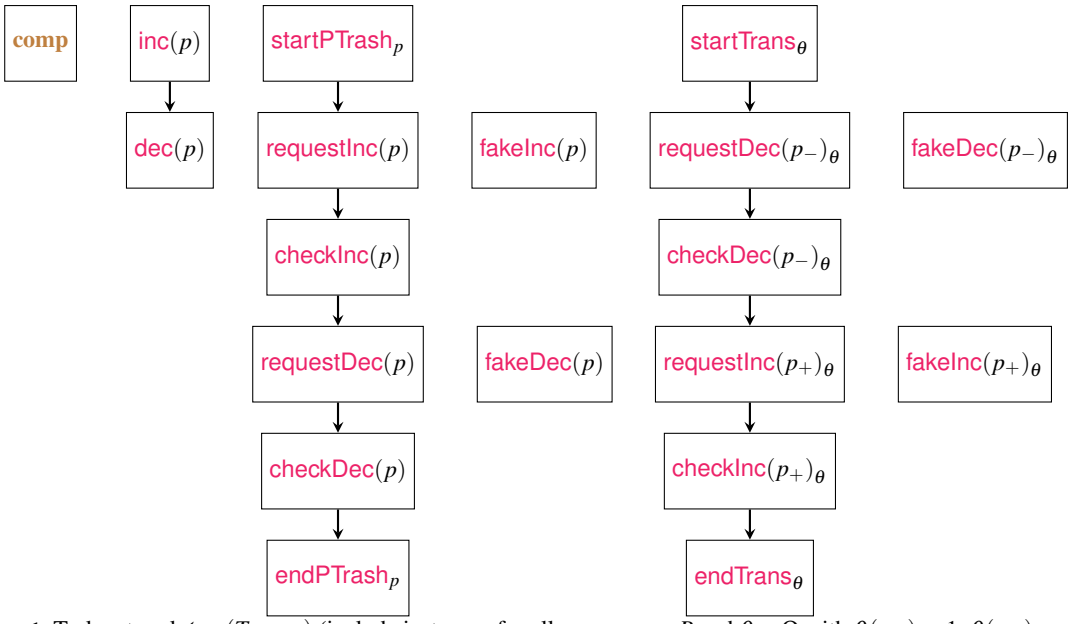

Figure 1: Task network $\mathfrak{t} = (T, \prec, \alpha)$ (include instances for all $p, p_-, p_+ \in P$ and $\theta \in \Theta$ with $\theta(p_+) = 1$, $\theta(p_-) = -1$).

Table 1: Action mapping $\delta(\mathsf{pr}) = (\pi_+, \pi_-, e_+, e_-)$.

pr	π_+	π_-	e_+	e_-
startPTrashPhase	searchPhase	transInProg	pTrashPhase	searchPhase
startRTrashPhase	pTrashPhase	pTrashInProg	rTrashPhase	pTrashPhase
startPTrash		searchPhase pTrashInProg	pTrashInProg	
endPTrash				pTrashInProg
startTrans		pTrashPhase transInProg	transInProg	
endTrans				transInProg
inc(p)	inc(p)	rTrashPhase		inc(p)
dec(p)	dec(p)	rTrashPhase		dec(p)
requestInc(p)		inc(p)	inc(p)	
checkInc(p)		inc(p)		
requestDec(p)		dec(p)	dec(p)	
checkDec(p)		dec(p)		
fakeInc(p)	rTrashPhase			inc(p)
fakeDec(p)	rTrashPhase			dec(p)

Table 2: Example instance of PETRI_1.

	p	q
θ	1	0
η	0	1
τ_0	0	0
τ_1	1	0

1 cont, stop,
2 $\mathsf{startTrans}_\theta, \mathsf{requestInc}(p)_\theta, \mathsf{inc}(p), \mathsf{checkInc}(p)_\theta, \mathsf{endTrans}_\theta$,
3 $\mathsf{requestDec}(p), \mathsf{dec}(p), \underline{\mathsf{checkDec}(p)}$,
4 $\underline{\mathsf{startPTrashPhase}}$,
5 $\mathsf{startPTrash}_q, \mathsf{requestInc}(q), \mathsf{inc}(q), \mathsf{checkInc}(q)$,
6 $\mathsf{requestDec}(q), \mathsf{dec}(q), \mathsf{checkDec}(q), \mathsf{endPTrash}_q$,
7 $\underline{\mathsf{startRTrashPhase}}$,
8 $\mathsf{startPTrash}_p, \mathsf{requestInc}(p), \mathsf{fakeInc}(p), \mathsf{checkInc}(p)$,
9 $\mathsf{requestDec}(p), \mathsf{fakeDec}(p), \mathsf{checkDec}(p), \mathsf{endPTrash}_p$,
10 $\mathsf{fakeInc}(q), \mathsf{fakeDec}(q)$,
11 $\mathsf{startTrans}_\eta, \mathsf{requestInc}(q)_\eta, \mathsf{fakeInc}(q)_\eta, \mathsf{checkInc}(q)_\eta, \mathsf{endTrans}_\eta$,
12 $\mathsf{fakeInc}(p)_\theta$

Figure 2: Example solution to Π ($\underline{\text{tasks in } T_0}$).

plan starts with (4) and the searchPhase ends with (5) (both accompanied by executions of token tasks). In the interim, executions of transition tasks simulate transition firings. Firing a transition θ corresponds to executing an entire chain of transition tasks of θ, also accompanied by executions of token tasks. The chain features tasks for each place that has nonzero value under θ, ensuring that the Petri net state encoded by the HTN state is updated appropriately. Line 2 of Fig. 2 exemplifies this process. 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 copy of some $\mathsf{startTrans}_{\theta'}$ can be executed. Moreover, as transInProg is a negative precondition of startPTrashPhase, every transition firing simulation commenced in searchPhase has to be finished in searchPhase. After the boundary tasks (4) are completed, the task network encodes τ_0. The task network encodes τ_1 before the excution of (5) iff the task network encodes the zero state of $\mathcal{N}$ after the execution of (5). Hence the transition firings corresponding to the transition tasks executed during searchPhase constitute a path through $\Phi_\mathcal{N}$ from τ_0 to τ_1 iff the task network encodes the zero state of $\mathcal{N}$ at the end of searchPhase. Hence the crucial claim is that an HTN state without remaining boundary tasks and with propositional state $\{\mathtt{searchPhase}\}$ encodes the zero state of $\mathcal{N}$ iff executing startPTrashPhase allows one to reach $\mathfrak{o}$ in Φ_Π.

In pTrashPhase, only token tasks and "place trashing tasks" (third column of Fig. 1) can be executed. Executing an entire chain of place trashing tasks of $p \in P$ needs to be accompanied by the execution of exactly one $\mathsf{inc}(p)$ task and exactly one $\mathsf{dec}(p)$ task. For an example, see lines 5-6 in Fig. 2. Similar to transInProg, the variable pTrashInProg prevents that we work on two place trashings at the same time, and every place trashing commenced in pTrashPhase has to be finished in pTrashPhase. Hence for each $p \in P$, equally many copies of $\mathsf{inc}(p)$ as $\mathsf{dec}(p)$ are executed in this phase, so (since they cannot be executed in the final rTrashPhase phase) the HTN state must encode the zero state of $\mathcal{N}$ when executing startPTrashPhase. Conversely, we have to show that after place trashing all leftover token tasks, all remaining tasks can be finished starting with startRTrashPhase.

To this end, tasks with names $\mathsf{fakeInc}(p)$ and $\mathsf{fakeDec}(p)$ can be executed in rTrashPhase. As there are enough such tasks, this effectively means that incrementation and decrementation commands can be ignored. Hence all remaining place trashing tasks and transition tasks can be executed (lines 8-9 respectively 11 in Fig. 2). □

The method task network $\mathfrak{t}$ in Fig. 1 consists of *parallel sequences*. This type of structure occurs frequently in HTN planning; cf. [7].

4.3 Main result

Theorem 9. *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* ACKERMANN*-complete.*

Proof. Thm. 3 and Props. 7 and 8. □

5 Numeric planning

While the main contribution of this paper is a new class of barely decidable HTN planning problems, we also identified a new barely decidable fragment of numeric planning. While HTN planning and numeric planning have not been regarded as related formalisms in the past, we nevertheless think that both barely decidable fragments share significant similarities – as both relate to Petri nets.

Like HTN planning, numeric planning is an extension of classical planning. In contrast to HTN planning, classical planning features only primitive tasks, and no methods or compound tasks (see e.g. STRIPS [14]). A classical plan π is a sequence of primitive tasks (called actions in this context) that is executable and for which the state reached after executing the plan satisfies a given goal condition.

Numeric planning extends classical planning by allowing for numbers in the state representation. Instead of propositional variables, numeric planning features a set V of *numeric variables*. A state is any function $V \mapsto \mathbb{Z}$, i.e., any assignment of integer values to these variables [18, 38]. In general, we can allow for values in $\mathbb{Q}$, but for the setting we will consider later on, this is as expressive as restricting to $\mathbb{Z}$ only (see [38]). Preconditions and effects of actions are numeric conditions and numeric changes, respectively. A precondition is of the form $f \bowtie 0$ where f is an integer polynomial over V and $\bowtie$ is any of the relations $=, \neq, \geq$. Similarly, effects are of the form $v = f$ where $v \in V$, indicating that the value of variable v is changed to the result of evaluating f in the state. The goal G is a set of preconditions.

In the just presented form, numeric planning is undecidable [18]. Research has thus focussed on identifying restrictions to the preconditions and effects under which the plan existence problem is still decidable. A commonly studied restriction is *simple numeric planning*, where effects are restricted to be of the form $v = v \pm c$ for $v \in V$ and $c \in \mathbb{Z}$ only [21, 35, 38]. We will abbreviate $v = v \pm c$ as $v \pm\!= c$. Helmert [18] proved that already a subclass of simple numeric planning is undecidable, namely if we restrict preconditions to be of the form $v \bowtie 0$ and effects to be $v \pm\!= 1$. Plan existence for simple numeric planning becomes decidable in two known syntactic cases $(\mathscr{C}_p, \mathscr{C}_\emptyset, \mathscr{E}^{=c}_{\pm c})$ and $(\mathscr{C}_p, \mathscr{C}_p, \mathscr{E}^{=c}_{+c})$ in Helmert's notation [18]. The first option restricts all preconditions to be empty and the goals to be of the form $f(v) \bowtie 0$ where f is a polynomial over a single state variable. The other restricts effects to the form $v \mathrel{+}= c$ for $c \in \mathbb{N}$, and both preconditions and goals to $f(v) \bowtie 0$. In both cases, effects of the type $v = c$ for $c \in \mathbb{Z}$ can also be allowed. Both classes are rather restrictive: in the first, we cannot have numeric preconditions, while in the second, the value of a variable can only increase or be set to a fixed value, so never decreases by a constant. In another direction, Shleyfman et al. [38] studied structural restrictions to the causal graph [19] of the problem and the special case of only a single numeric variable.

We identified a new class of decidable numeric planning problems that have not yet been covered by any of the previous studies: Positive Numeric Planning (PNP). In PNP, we enforce that the value of every numeric state variable is non-negative and impose further restrictions. A PNP problem (V, A, I, G) is defined as follows. A state s is a function $V \to \mathbb{N}$, i.e., it maps each variable $v \in V$ to a non-negative integer $s(v)$. The initial state I is a state. We denote the set of actions as A. Actions a are defined by $\langle pre, eff \rangle$, where *pre* and *eff* are sets of numeric conditions or numeric effects, respectively. Preconditions are of the form $v \geq c$ for $c \in \mathbb{N}$. Effects are of the form $v \pm\!= c$ for $c \in \mathbb{N}$. Further, if $(v \mathrel{-}= c) \in eff$, then we require that $(v \geq c') \in pre$ for some $c' \geq c$. This condition ensures that in any reachable state, the value of all variables is non-negative – and is the *novel restriction* of PNP. The goal is a set of conditions that can be of the form $v = c$ or $v \geq c$ for constants $c \in \mathbb{N}$. We consider two subfragments: $\mathrm{PNP}_\geq$ and $\mathrm{PNP}_=$ where only goals of the form $v \geq c$ or $v = c$, respectively, are allowed.

PNP is a special case of simple numeric planning. In contrast to previously known decidable classes, it allows for a combination of decreasing numeric effects $(v \mathrel{-}= c)$ and some numeric preconditions. If we were to allow preconditions of the form $v = c$, we would

obtain an undecidable problem (as $(\mathscr{C}_c, \mathscr{C}_c, \mathscr{E}_{\pm 1})$ in Helmert's notation [18] is undecidable). As such, PNP is a new island of decidability in numeric planning.

On the other hand, PNP problems are also extremely hard problems – they are ACKERMANN-complete.

Proposition 10. *PLANEX(PNP) reduces to PETRI.*

Proof. Consider a PNP problem $\Pi = (V, A, I, G)$. We set $P = V \sqcup A \sqcup \{\mathfrak{C}, \mathfrak{G}\}$, i.e., the places of the Petri net are the numeric variables, the actions, and two new places: $\mathfrak{C}$ is a place for general execution control and $\mathfrak{G}$ signifies that the goal has been reached. Consider an action $a = \langle pre, eff \rangle \in A$. We can assume w.l.o.g. that a has an effect related to $v \in V$ iff a has a precondition on v; if only has an effect, we can add the precondition $v \geq 0$, while if it only has a precondition, we can add the effect $v \mathrel{+}= 0$, without changing the semantics of Π. Define two transitions $\theta_a^-, \theta_a^+ : P \to \mathbb{Z}$ by

$$\theta_a^-(p) = \begin{cases} -c & \big((p \geq c) \in pre\big) \\ 1 & (p = a) \\ -1 & (p = \mathfrak{C}) \\ 0 & (\text{otherwise}) \end{cases}$$

and

$$\theta_a^+(p) = \begin{cases} c_1 + c_2 & \big((p \geq c_1) \in pre \;\&\; (p \mathrel{+}= c_2) \in eff\big) \\ c_1 - c_2 & \big((p \geq c_1) \in pre \;\&\; (p \mathrel{-}= c_2) \in eff\big) \\ -1 & (p = a) \\ 1 & (p = \mathfrak{C}) \\ 0 & (\text{otherwise}). \end{cases}$$

These are well-defined as the action a cannot have multiple preconditions or multiple effects on the same state variable. Note that by the definition of PNP, we have that $c_1 - c_2 \geq 0$ in the second case of the definition of θ_a^+. Executing action a corresponds to firing θ_a^- followed by firing θ_a^+.

We then add a transition θ^G with

$$\theta^G(p) = \begin{cases} -c & \big((p \geq c) \in G\big) \\ -c & \big((p = c) \in G\big) \\ -1 & (p = \mathfrak{C}) \\ 1 & (p = \mathfrak{G}) \\ 0 & (\text{otherwise}). \end{cases}$$

Lastly, for every variable $v \in V$ such that there is no goal condition of the form $(v = c) \in G$, we add a transition θ_v with

$$\theta_v(p) = \begin{cases} -1 & (p = v) \\ 0 & (\text{otherwise}). \end{cases}$$

Note that it is correct to allow transitions θ_v to fire at any time as the firing sequence can always be re-ordered s.t. θ_v fire only after θ^G. We are then asking whether from the state $\tau_0 \in \Omega_{\mathscr{N}}$ defined by

$$\tau_0(p) = \begin{cases} I(p) & (p \in V) \\ 1 & (p = \mathfrak{C}) \\ 0 & (\text{otherwise}) \end{cases}$$

we can reach the state τ_1 defined by

$$\tau_1(p) = \begin{cases} 1 & (p = \mathfrak{G}) \\ 0 & (\text{otherwise}). \end{cases}$$

By construction, this is the case iff Π is solvable. □

Proposition 11. *PETRI reduces to PLANEX(PNP$_=$).*

Proof. Let $\mathscr{N} = (P, \Theta)$ be a Petri net and τ_0, τ_1 two states. We then set $V = P$, $I = \tau_0$, $G = \{p = \tau_1(p) \mid p \in P\}$, and $A = \{a_\theta : \theta \in \Theta\}$, where a_θ has preconditions $v \geq -\theta(p)$ and effects $v \mathrel{-}= -\theta(p)$ for all $p \in P$ with $\theta(p) < 0$ and effects $p \mathrel{+}= \theta(p)$ for all $p \in P$ with $\theta(p) > 0$. A plan for (V, A, I, G) implies the existence of a firing sequence reaching τ_1 and vice versa. □

Theorem 12. *PLANEX(PNP) and PLANEX(PNP$_=$) are ACKERMANN-complete.*

Proof. Thm. 3 and Props. 10 and 11. □

Interestingly, the problem PLANEX(PNP$_\geq$) has also not been studied in the literature and is much easier. For this, we use the coverability problem of Petri nets PETRICOV. Given a Petri net $\mathscr{N} = (P, \Theta)$ and two states τ_0 and τ_1, we have to decide whether there is state τ_1' reachable from τ_0 that is point-wise greater or equal to τ_1, i.e., $\forall p \in P : \tau_1(p) \leq \tau_1'(p)$. PETRICOV is EXPSPACE-complete [26, 32]. Using the same reduction as in Prop. 11 but for PETRICOV, we obtain EXPSPACE-hardness for PLANEX(PNP$_\geq$). For membership, we can use the same construction as in Prop. 10, but have to ask whether the state τ_1 is covered from τ_0. We thus obtain:

Corollary 13. *PLANEX(PNP$_\geq$) is EXPSPACE-complete.*

Lastly, there is an extension to Petri nets: Petri nets with *inhibitor arcs* [8]. Inhibitor arcs are additional conditions of the form (θ, p) for transitions θ and places p. The transition θ of an inhibitor arc can only fire if the place p currently contains zero tokens. Reachability for Petri nets with one inhibitor arc is decidable [33], but undecidable for at least two inhibitor arcs [28]. The exact complexity of the former problems is not known. From these results, it follows that if we add one $v = 0$ precondition to a PNP problem, we still retain decidability, while the problem becomes undecidable as soon as we allow for two such preconditions.

6 Conclusion and future work

We introduced several new fragments of HTN and numeric planning and proved that many of them are complete for the large class ACKERMANN of decidable problems. Table 3 lists some natural fragments. Notice that the new fragments have higher computational complexity than the previously known ones.

Table 3: Some fragments of planning and their complexities.

Fragment $\mathscr{Q}$	Complexity of PLANEX($\mathscr{Q}$)	Reference
All HTN problems	Undecidable	[11]
$\mathscr{H}_1$ (one-hole-digging)	ACKERMANN-complete	Thm. 9
$\mathscr{I}$ (initial)	ACKERMANN-complete	Thm. 9
$\mathscr{F}$ (final)	ACKERMANN-complete	Thm. 9
$\mathscr{C}$ (clean)	ACKERMANN-complete	Thm. 9
Tail-recursive	EXPSPACE-complete	[4]
Acyclic	NEXPTIME-complete	[4]
Total order	EXPTIME-complete	[4]
Regular	PSPACE-complete	[11]
Simple numeric	Undecidable	[18]
PNP (positive numeric)	ACKERMANN-complete	Thm. 12
PNP$_=$	ACKERMANN-complete	Thm. 12
PNP$_\geq$	EXPSPACE-complete	Corol. 13
$(\mathscr{C}_p, \mathscr{C}_\emptyset, \mathscr{E}_{\pm c}^{=c})$	PSPACE-complete	[18]
$(\mathscr{C}_p, \mathscr{C}_p, \mathscr{E}_{+c}^{=c})$	PSPACE-complete	[18]

The proof of Thm. 9 relies on the recently established 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 Prop. 7 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 [24]. However, is PLANEX($\mathscr{H}_2 \cap \mathscr{L}$) decidable?

References

[1] R. Alford. *Search complexities for HTN planning*. PhD thesis, 2013.

[2] R. Alford, U. Kuter, and D. Nau. 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)*, pages 1629–1634. AAAI Press, 2009.

[3] R. Alford, V. Shivashankar, U. Kuter, and D. S. Nau. HTN problem spaces: Structure, algorithms, termination. In *Proceedings of the 5th Annual Symposium on Combinatorial Search (SoCS 2012)*, pages 2–9. AAAI Press, 2012.

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

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

[6] G. Behnke, D. Höller, and S. Biundo. totSAT – Totally-ordered hierarchical planning through SAT. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, pages 6110–6118. AAAI Press, 2018.

[7] G. Behnke, F. Pollitt, D. Höller, P. Bercher, and R. Alford. Making translations to classical planning competitive with other HTN planners. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, pages 9687–9697. AAAI Press, 2022. doi: 10.1609/aaai.v36i9.21203.

[8] H. K. Büning, T. Lettmann, and E. W. Mayr. Projections of vector addition system reachability sets are semilinear. *Theoretical computer science*, 64(3):343–350, 1989.

[9] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1):165–204, 1994.

[10] W. Czerwiński and L. Orlikowski. Reachability in vector addition systems is ackermann-complete. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1229–1240, 2022. doi: 10.1109/FOCS52979.2021.00120.

[11] K. Erol, J. Hendler, and D. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, 1994.

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

[13] J. Esparza. Decidability and complexity of petri net problems – an introduction. *Lecture Notes in Computer Science*, 1491:374–428, 1998.

[14] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4): 189–208, 1971.

[15] B. Gazen and C. Knoblock. 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, pages 221–233. Springer, 1997.

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

[17] M. Hack. Decision problems for petri nets and vector addition systems. *Project MAC technical report*, 59, 1975.

[18] M. Helmert. Decidability and undecidability results for planning with numerical state variables. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002)*, pages 44–53, 2002.

[19] M. Helmert. A planning heuristic based on causal graph analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pages 161–170, 2004.

[20] S. Hickmott, J. Rintanen, S. Thiébaux, and L. White. Planning via petri net unfolding. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1904–1911, 2007.

[21] J. Hoffmann. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of artificial intelligence research*, 20:291–341, 2003.

[22] D. Höller. 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)*, pages 159–167. AAAI Press, 2021.

[23] D. Höller, G. Behnke, P. Bercher, and S. Biundo. Language classification of hierarchical planning problems. In *Proceedings of the 21st*

European Conference on Artificial Intelligence (ECAI 2014), volume 263, pages 447–452. IOS Press, 2014.

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

[25] J. Leroux and S. Schmitz. Reachability in vector addition systems is priive-recursive in fixed dimension. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*, number 50, pages 1–13, 2019.

[26] R. Lipton. The reachability problem requires exponential space, 1976.

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

[28] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1971.

[29] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, 1989.

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

[31] I. Pohl. Bi-directional and heuristic search in path problems, 1969.

[32] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

[33] K. Reinhardt. Reachability in petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008.

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

[35] E. Scala, P. Haslum, S. Thiébaux, and M. Ramirez. Subgoaling techniques for satisficing and optimal numeric planning. *Journal of Artificial Intelligence Research*, 68:691–752, 2020.

[36] S. Schmitz. Complexity hierarchies beyond elementary. *ACM Transactions on Computation Theory*, 8(1), 2016. ISSN 1942-3454. doi: 10.1145/2858784.

[37] D. Schreiber. 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)*, 2021.

[38] A. Shleyfman, D. Gnad, and P. Jonsson. Structurally restricted fragments of numeric planning–a complexity analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12112–12119, 2023.