How Good is Perfect?
On the Incompleteness of A* for Total-Order HTN Planning

Mohammad Yousefi[1], Mario Schmautz[2], Patrik Haslum[1], Pascal Bercher[1]

[1]School of Computing, The Australian National University, Canberra, Australia
[2]Independent Researcher*
mohammad.yousefi@anu.edu.au, mschmautz@proton.me, patrik.haslum@anu.edu.au, pascal.bercher@anu.edu.au

Abstract

This paper reveals the inherent limitations of A* in HTN planning by identifying various cycle types induced by the task hierarchy and analyzing their effects on the termination of the algorithm. We prove that A* even with the perfect heuristic, and for the special case of totally ordered problems, which are known to be decidable, is incomplete. An especially interesting results is that having a visited list (i.e., graph search) with the null heuristic has better termination guarantees than tree search with the perfect heuristic. We provide a polynomial-time test for detecting those cycles that render A* incomplete, and analyzed all existing benchmark domains from the most-recent international planning competition. Results show that in more than half of all domains, A* tree search would be incomplete even with the perfect heuristic, and in roughly 40% of cases A* graph search might also be incomplete depending on the provided heuristic function. We also point to a normal form that preserves semantics and guarantees completeness of the resulting models, though implementation and testing remains for future work.

Introduction

Hierarchical Task Network (HTN) planning is a framework that allows decomposing abstract tasks into smaller subtasks (Bercher, Alford, and Höller 2019; Ghallab, Nau, and Traverso 2016). This mechanism results in interesting deviations from classical planning as HTN planning is domain configurable (Nau et al. 2003), undecidable (Geier and Bercher 2011; Erol, Hendler, and Nau 1996) and more expressive (Lin and Bercher 2022; Höller et al. 2016;2014). Decomposition is fundamentally a zero-cost operation since it only changes the problem representation. This, by itself, violates the assumptions of the A* search algorithm (Pearl 1984; Hart, Nilsson, and Raphael 1968), but it has been shown that as long as zero-cost transitions cannot be repeated infinitely many times in a row, the algorithm is still complete, albeit with a cost to performance (Aghighi and Backstrom 2016; Benton et al. 2010).

In this paper, we show that in hierarchical planning, due to the recursive nature of decompositions, infinite zero-cost paths are rampant even in severely restricted subclasses. Furthermore, unlike classical planning where almost perfect heuristics cause performance issues (Helmert and Röger 2008), we prove that A* in totally-ordered HTN planning problems (which is known to be decidable (Erol, Hendler, and Nau 1996)) is *incomplete* for all almost perfect heuristics. Even with the perfect heuristic, the completeness is conditioned on the tie-breaking strategy. This theoretical result is particularly significant given A*'s central role in recent HTN planning systems. Notably, the winner of all three (agile, satisficing, and optimal) Totally Ordered HTN (TOHTN) planning tracks of the IPC 2023 utilized A* search (Olz, Höller, and Bercher 2023) with additional pruning techniques (Olz and Bercher 2023). Even more, the winner of the optimal partial order track also utilized A* (Höller 2023).

Since our analysis focuses on a subclass of HTN planning (i.e., TOHTN planning), the findings also extend to the general case of partially ordered problems. By examining A*'s limitations in this important decidable fragment, we illustrate fundamental challenges in deploying heuristic search for HTN planning. Our contributions are as follows:

- We prove that, even with the perfect heuristic, A* is incomplete for HTN planning,
- characterize the conditions under which the search process may not terminate, and provide a poly-time algorithm to detect them prior to search,
- provide a solution-preserving transformation of TOHTN domains that guarantees the completeness of A* in the transformed domain, and
- conduct an evaluation of totally-ordered hierarchical domains in the International Planning Competition (IPC) 2023 to see how frequently such conditions happen in practical scenarios.

Formalization

HTN Planning

HTN planning is concerned with performing tasks, as opposed to achieving a particular goal. As such, there is no explicit goal description. However, if necessary, one can compile a goal description to the precondition of a new primitive task that must be executed as the last step (Geier and Bercher 2011). A TOHTN planning problem is a special case of HTN

*Some contributions were made while still affiliated with the Institute of Artificial Intelligence at the Ulm University, Germany.

planning where all task networks are restricted to be sequential. Our definitions are based on the formalization provided by Behnke, Höller, and Biundo (2018). We use X^* to denote the Kleene closure of the set X (Hopcroft, Motwani, and Ullman 2006).

Definition 1 (TOHTN Planning Problem). *A TOHTN planning problem is a tuple $P = \langle F, C, A, M, c_I, s_I \rangle$ where:*

- *F is a finite set of facts,*
- *C, A are disjoint finite sets of compound and primitive task names ($T = C \cup A$), respectively,*
- *$M \subseteq C \times T^*$ is a finite set of totally ordered decomposition methods,*
- *$c_I \in C$ is the initial compound task name,*
- *$s \in 2^F$ is the initial state.*

A task network is a finite string of compound and primitive task names. For this reason, we use the notation $\overline{t_1 \dots t_k}$ to denote a string (task network) of symbols (tasks). Furthermore, by abuse of notation, $t_j \in \overline{t_1 \dots t_k}$ is used to denote that a task t_j appears in task network $\overline{t_1 \dots t_k}$.

Definition 2 (Task Network). *A (totally ordered) task network is defined as $tn = \overline{t_1 \dots t_k}$ where $tn \in T^*$.*

For each primitive task name, $a \in A$, its action is a tuple $\langle pre(a), add(a), del(a) \rangle$[1] where $pre(a) \subseteq F$ denotes the preconditions that must hold in order to execute the action, and $add(a), del(a) \subseteq F$ denote the add and delete effects of a, respectively. An action $a \in A$ is *executable* in state $s \in 2^F$ iff $pre(a) \subseteq s$. The state transition function $\gamma \colon A \times 2^F \to 2^F$ is defined as:

$$\gamma(a, s) = \begin{cases} (s \setminus del(a)) \cup add(a) & \text{if } a \text{ is executable in } s \\ \textit{undefined} & \text{otherwise} \end{cases}$$

In contrast, compound tasks are decomposed to smaller subtasks. As mentioned, the cost of decomposing a compound task is always zero.

Definition 3 (Decomposition). *Given a task network $tn_1 = \overline{t_1 c t_2} \in T^*$ where $c \in C$, a method $m = \langle c, \omega \rangle \in M$ substitutes c with ω, denoted as $tn_1 \to^c_m tn_2$, resulting in a new task network $tn_2 = \overline{t_1 \omega t_2}$.*

Given the definitions of action execution and decomposition, we can now define the progressions of a task network (Höller et al. 2020). Similar to decomposition, we denote the progression of a task network tn to one of its successors, tn', by $tn \overset{p}{\rightsquigarrow} tn'$ where p is the name of the progressed task.

Definition 4 (Progression). *Let $s \in 2^F$ be a state, and $tn = \overline{t_1 \cdots t_k}$ be a task network ($k \geq 1$). The set of successors of tn under progression, denoted as Prog(tn), is defined as:*

- *if $t_1 \in A$:*
  - *if t_1 is executable in s, then Prog$(tn) = \{\overline{t_2 \dots t_k}\}$,*
  - *otherwise, Prog$(tn) = \emptyset$.*
- *if $t_1 \in C$:*
  - *Prog$(tn) = \{\overline{\alpha t_2 \dots t_k} \mid \langle t_1, \alpha \rangle \in M\}$*

[1]For brevity's sake we have omitted the explicit mapping function $\delta \colon A \to 2^F \times 2^F \times 2^F$ known from other formalizations (Bercher, Alford, and Höller 2019).

The goal of HTN planning is to refine the initial compound task to the empty task network (i.e., perform all the tasks), which we denote as ϵ.

Definition 5 (Solution). *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN planning problem. A (possibly empty) sequence of progression $\pi = \overline{c_I} \overset{p_1}{\rightsquigarrow} tn_2 \overset{p_2}{\rightsquigarrow} \cdots \overset{p_m}{\rightsquigarrow} \epsilon$ is a solution to P iff there exists a sequence of states $s_1, s_2, \dots, s_{m+1}$ with $s_1 = s_I$ such that for all $i \in \{1, \dots m\}$, we have:*

- *if progression p_i is an action execution, then p_i is executable in s_i and $s_{i+1} = \gamma(p_i, s_i)$,*
- *if progression p_i is a decomposition, then $s_{i+1} = s_i$.*

π is an optimal solution iff it is minimal with respect to number of primitive action progressions.

We use context-free grammar notation $E \to \overline{t_1 \dots t_k} \mid \dots \mid \overline{t'_1 \dots t'_j}$ to represent that a compound task, E, can be decomposed into different task networks, separated by "|", without mentioning the methods. We use upper cases letters (e.g. E), lower case letters (e.g., x), and ϵ to denote compound tasks, primitive tasks, and empty methods (i.e., a method with zero subtasks), respectively. For example, $E \to TT \mid x \mid \epsilon$ denotes a compound task E which can be decomposed either into two instances of compound task T, a primitive action x, or the empty task network (i.e., removed).

A* Search

A* is a best-first search algorithm that, usually, operates on an implicit graph to find the minimum cost path from an initial node to a goal one by iteratively expanding nodes based on the sum of accumulated cost and the estimated remaining effort (Nilsson 1982). Our implementation of A* (Alg. 1) adheres to the textbook definition (Edelkamp, Schroedl, and Koenig 2010). However, since all presented heuristics are consistent, we have intentionally omitted fringe reopening. Furthermore, we consider the tree search version as well (i.e., A* without a *CLOSED* list) because establishing whether two task networks are isomorphic or not is GI-Complete in the general case of partially-ordered problems (Behnke, Höller, and Biundo 2015). To define the search formally, we adopt the search node definition in HTN progression space (Höller et al. 2020), and denote the set of all search nodes as $\mathbb{SN}$.

Definition 6 (Search Node). *A search node is a tuple $SN = \langle \Gamma, tn, s \rangle$ where:*

- *$\Gamma = a_1, a_2, \dots, a_n$ is a, possibly empty, sequence (with possible repetition) of actions that have been executed so far to reach this search node,*
- *tn is the remaining task network,*
- *s is the current state,*

Two search nodes $SN_1 = \langle \Gamma_1, tn_1, s_1 \rangle$ and $SN_2 = \langle \Gamma_2, tn_2, s_2 \rangle$ are isomorphic iff we have $tn_1 = tn_2$ and $s_1 = s_2$.

For all search nodes, we define the *f-value* to be the sum of the accumulated cost (also referred to as the g-value) and an estimation of the remaining cost (h-value). Note that since decompositions are zero-cost transitions, they do not alter

Algorithm 1: A* Search

Input: cost function w, heuristic h, successor generation function *Expand*, implicit problem graph start node s, and goal test *Goal*.
Output: Cost-optimal path from s to a goal node, or $\emptyset$ if no such path exists

```
CLOSED ← ∅;
OPEN ← {s};
f(s) ← h(s);
while OPEN ≠ ∅ do
    Remove u from OPEN with minimum f(u);
    Insert u into CLOSED;
    if Goal(u) then return Path(u);
    else
        Succ(u) ← Expand(u);
        foreach v ∈ Succ(u) do
            if v ∈ OPEN then
                if g(u) + w(u,v) < g(v) then
                    parent(v) ← u;
                    f(v) ← g(u) + w(u,v) + h(v);
            else if v ∈ CLOSED then
                if g(u) + w(u,v) < g(v) then
                    parent(v) ← u;
                    f(v) ← g(u) + w(u,v) + h(v);
            else
                parent(v) ← u;
                f(v) ← g(u) + w(u,v) + h(v);
                Insert v into OPEN with f(v);
return ∅
```

the g-value. Formally, for a search node $SN = \langle \Gamma, tn, s \rangle$, we have $f(SN) = g(SN) + h(SN)$ where $g(SN) = |\Gamma|$ and $h(SN)$ is obtained from a heuristic function $h\colon\ T^* \times 2^F \rightarrow \mathbb{N}_{\geq 0} \cup \{\infty\}$. The perfect heuristic function returns the exact remaining cost for all search nodes.

Definition 7 (Perfect Heuristic)**.** *Let $g^*(tn, s)$ be the minimum cost to refine tn into a solution from state s. The perfect heuristic function, h^*, is defined as follows:*

$$h^*(tn, s) = \begin{cases} g^*(tn, s) & \text{if solution exists} \\ \infty & \text{otherwise} \end{cases}$$

The almost perfect heuristic is always, by a constant amount, less than the optimal solution cost (Helmert and Röger 2008).

Definition 8 (Almost Perfect Heuristic)**.** *Let $\Delta \in \mathbb{N}_{>0}$ be a small positive constant. The almost perfect heuristic function, h^Δ, is defined as follows:*

$$h^\Delta(tn, s) = \begin{cases} max(h^*(tn, s) - \Delta, 0) & \text{if solution exists} \\ \infty & \text{otherwise} \end{cases}$$

On the other end of the spectrum is the null heuristic which adds no information.

Definition 9 (Null Heuristic)**.** *For all task networks tn and states s, the null heuristic is defined as $h_0(tn, s) = 0$.*

Incompleteness

In this section, we show that the search space of many TOHTN planning problems contain an infinite zero-cost path, a sufficient condition for the incompleteness of A* (Benton et al. 2010). Importantly, our result remains valid even in the absence of zero-cost actions, as our construction depends solely on decomposition transitions and is unaffected by action costs. We start by introducing the notion of a cycle in the context of TOHTN planning. Existence of a cycle is a necessary condition to have an infinite search space (Alford et al. 2012).

Definition 10 (Cycle)**.** *Let $\sigma = \overline{c_1} \rightarrow^{c_1}_{m_1} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ be a sequence of decompositions ($k \geq 1$). σ is an (unrestricted) cycle iff $tn_{k+1} = \overline{\alpha c_1 \beta}$ where $\alpha, \beta \in T^*$. We refer to c_1 as the initiator of the cycle.*

As a simple example, consider $C \rightarrow xC$ which forms a cycle with a single decomposition. Next, we restrict this definition to be zero-cost i.e., no primitive action is executed in this cycle to change the g-value. This cycle is akin to left-recursion in formal grammars.

Definition 11 (ϵ-prefix Cycle)**.** *Let $\sigma = tn_1 \rightarrow^{c_1}_{m_1} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ be a cycle. σ is an ϵ-prefix cycle iff $tn_{k+1} = \overline{c_1 \beta}$ where $\beta \in T^*$.*

The existence of an ϵ-prefix cycle in a problem is the sufficient condition to construct a *finite* zero-cost path in the search space. The reason for finiteness is that the suffix, β, may be empty. In that case, the cycle leads to what is known as an empty cycle (Behnke, Höller, and Biundo-Stephan 2019) where the starting task network is identical to the resulting task network.

Definition 12 (Empty Cycle)**.** *Let $\sigma = tn_1 \rightarrow^{c_1}_{m_1} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ be an ϵ-prefix cycle. σ is an empty cycle iff we have $tn_{k+1} = tn_1$.*

We now prove that the perfect heuristic cannot differentiate between the task networks that are on an empty cycle.

Theorem 1. *Let $\sigma = tn_1 \rightarrow^{c_1}_{m_1} tn_2 \rightarrow^{c_2}_{m_2} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ be an empty cycle, and s be an arbitrary state. It holds that $h^*(tn_1, s) = h^*(tn_2, s) = \cdots = h^*(tn_{k+1}, s)$.*

Proof. Let $g^*(tn_1, s)$ be the optimal cost of refining tn_1 into a solution from s. From the empty cycle assumption, it follows that $h^*(tn_1, s) = g^*(tn_1, s) = g^*(tn_{k+1}, s)$. Additionally, the optimal cost of task networks $tn_2, \ldots, tn_k$ cannot be greater than $g^*(tn_1, s)$ because there exists a sequence of zero-cost operations to make each of them identical to tn_1 (i.e., decomposition to tn_{k+1}). Furthermore, the optimal cost of task networks $tn_2, \ldots, tn_k$ cannot be less than $g^*(tn_1, s)$. Suppose that there exists a tn_i in the cycle ($2 \leq i \leq k$) such that $g^*(tn_i, s) < g^*(tn_1, s)$. Then, there exists a sequence of zero-cost operations to turn tn_1 into tn_i which has a lower value than $g^*(tn_1, s)$. This implies that $g^*(tn_1, s)$ is not the optimal solution cost, which is a contradiction. Thus, the optimal cost of refining each task network in an empty cycle to a solution is equal. Hence, the perfect heuristic, given the same state, returns the same value for all of them. □

The *CLOSED* list detects an empty cycle and prevents its repetition. However, variants of A* that do not utilise a *CLOSED* list, such as tree search A* and Iterative Deepening A* (IDA*) (Korf 1985), are incomplete when this condition is present. In order to prove this, we need to introduce the concept of *f-Reachability*. This is crucial because such a cycle might exist, but may be pruned by an informed heuristic long before initiation.

Definition 13 (f-Reachable)**.** *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN planning problem with an optimal solution cost of g^*. Let $c \in C$ be a compound task, and h be a heuristic function. We define c as f-Reachable under h iff there exists a sequence of progressions $\overline{c_I} \overset{p_1}{\rightsquigarrow} tn_2 \overset{p_2}{\rightsquigarrow} \cdots \overset{p_m}{\rightsquigarrow} tn_{m+1}$ and a sequence of states $s_I, s_2, \ldots, s_{m+1}$ such that:*

1. *tn_{m+1} is of the form $\overline{c\beta}$ with $\beta \in T^*$,*
2. *for all $i \in \{1, \ldots m\}$, we have: if progression p_i is an action execution, then p_i is executable in s_i and $s_{i+1} = \gamma(p_i, s_i)$; otherwise $s_{i+1} = s_i$,*
3. *the following equation holds:*

$$\underbrace{\sum_{i=1}^{m} cost(p_i)}_{\text{cost to reach } tn_{m+1}} + \underbrace{h(tn_{m+1}, s_{m+1})}_{\text{estimated cost from } tn_{m+1}} \leq g^*$$

Our first incompleteness result is established by showing that in the presence of an f-Reachable task that can initiate an empty cycle, A* without a *CLOSED* list may not terminate.

Theorem 2. *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN problem such that there exists a $c \in C$ where c is the initiator of an empty cycle and c is f-Reachable under h^*. Tree search A* with h^* is incomplete.*

Proof. Let g^* be the optimal solution cost to P. Since c is the initiator of an empty cycle, we know that there exists a sequence of search nodes $SN_1, \ldots, SN_{k+1}$ corresponding to the decompositions $tn_1 \rightarrow^{c_1}_{m_1} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ where $tn_1 = tn_{k+1}$. By Thm. 1 and the fact that decomposition neither changes the state nor the cost, it follows that $f(SN_1) = f(SN_2) = \cdots = f(SN_{k+1})$. Next, by the assumption that c is f-Reachable, we know that $f(SN_1) \leq g^*$. Thus, A* may eventually expand SN_1, and its successor SN_2 will be inserted to *OPEN*. The same conditions hold for SN_2, and this process can continue until the cycle is complete; i.e., SN_k is in the *OPEN* list. Since, there is no duplicate detection, the cycle can be repeated. Hence, the algorithm may not terminate with certain tie-breaking strategies. To see this, consider a tie-breaking strategy that favors decomposition over action execution. In this case, A* continuously seeks further decompositions which is always available from the initiated cycle. □

Next, we introduce the concept of a "growing" empty cycle to create an *infinite* zero-cost path. This cannot be detected with a *CLOSED* list because once it is completed, the resulting task network is not identical to its starting form.

Definition 14 (Growing ϵ-prefix Cycle)**.** *Let $\sigma = tn_1 \rightarrow^{c_1}_{m_1} tn_2 \rightarrow^{c_2}_{m_2} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ be an ϵ-prefix cycle. σ is a growing ϵ-prefix cycle iff $tn_{k+1} = \overline{c_1\beta}$ where $\beta \in T^*$ and $\beta \neq \epsilon$.*

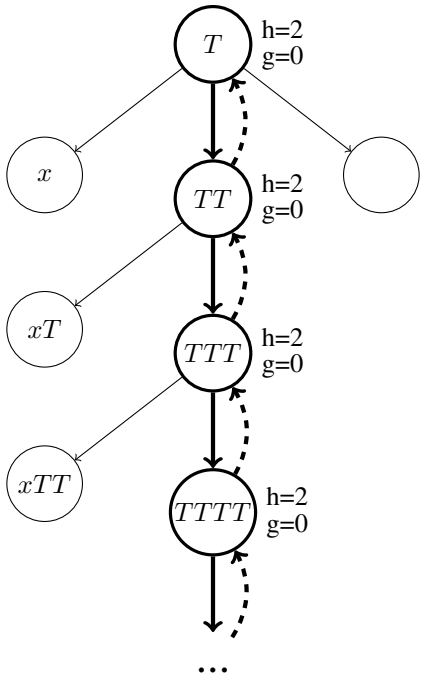

Figure 1: An illustration of how graph search A* gets stuck in a grow and shrink cycle for $T \rightarrow TT \mid x \mid \epsilon$. Nodes represent search nodes, and edges are possible transitions. For clarity, not all transitions are included. The thick arrows represent the grow part of a cycle, and the dashed thick arrows are the shrink part (empty methods). The heuristic value and the accumulated cost of each search node are noted using h and g, respectively.

Uniform cost search (i.e., A* search with the null heuristic) is incomplete when such a cycle is present.

Theorem 3. *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN problem such that there exists a $c \in C$ where c is the initiator of a growing ϵ-prefix cycle and c is f-Reachable under h_0 (i.e., uniform cost search). Graph search A* with h_0 is incomplete.*

Proof. Let g^* be the optimal solution cost to P. Since c is the initiator of a growing ϵ-prefix cycle, we know that there exists a sequence of search nodes $SN_1, \ldots, SN_{k+1}$ corresponding to the decompositions $tn_1 \rightarrow^{c_1}_{m_1} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ such that $tn_{k+1} = \overline{c_1\beta}$ where $\beta \in T^*$ and $\beta \neq \epsilon$ (Def. 14). Because the cycle is formed purely from decompositions, the g-values of all search nodes are equal. By the null heuristic assumption, it follows that their f-values are also equal. Since SN_1 is f-Reachable under the null heuristic, once it is expanded, the entire cycle may be generated as they have the same f-value. However, since $tn_{k+1} \neq tn_1$, tn_{k+1} will be inserted into the *OPEN* list. Thus, another cycle may be triggered again from tn_{k+1}. It follows that, given P, the algorithm may not terminate. □

While we have created an infinite zero-cost path using the

growing ϵ-prefix cycle, a mildly informed heuristic can still terminate the cycle as the estimated cost may increase. To see this, consider $T \rightarrow x \mid TT$. Even though there is a growing ϵ-prefix cycle and the g-value of the resulting search node does not differ from the starting one, the h-value may increase. Thus, the overall f-value is not the same. This implies that eventually this branch will lead to a search node with cost greater than g^* of the problem. As a last step, we need to further restrict this cycle to disable heuristic information. In order to do so, we need "shrinking", which allows the newly introduced tasks in a growing ϵ-prefix cycle to be removed from the final task network.

Definition 15 (Grow and Shrink Cycle)**.** *Let $\sigma = tn_1 \rightarrow^{c_1}_{m_1} tn_2 \rightarrow^{c_2}_{m_2} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ be a growing ϵ-prefix cycle. σ is a grow and shrink cycle iff in $tn_{k+1} = \overline{c_1\beta}$, it holds that $\beta \neq \epsilon$ and $\forall t \in \beta$, we have:*

1. *$t \in C$, and*
2. *there exists a sequence of decompositions $\bar{t} \rightarrow^{t}_{m_j} \cdots \rightarrow^{c_{j+n}}_{m_{j+n}} \epsilon$.*

The following corollary shows that even the perfect heuristic cannot see that the task networks on a grow and shrink cycle are growing.

Corollary 1. *Let $tn_1 \rightarrow^{c_1}_{m_1} \cdots \rightarrow^{c_k}_{m_k} tn_{k+1}$ be a grow and shrink cycle. There always exists a sequence of decompositions $tn_{k+1} \rightarrow^{c_{k+1}}_{m_{k+1}} \cdots \rightarrow^{c_{k+n}}_{m_{k+n}} tn_{k'}$ such that $tn_1 \rightarrow^{c_1}_{m_1} \cdots \rightarrow^{c_{k+n}}_{m_{k+n}} tn_{k'}$ forms an empty cycle.*

Proof. From Def. 15, we know that every task introduced in the grow cycle can be turned into ϵ. It follows that there exists a sequence of decompositions for each introduced task in tn_{k+1} that removes it, and these sequences can be chained together to transform tn_k to $tn_{k'} = tn_1$ in which the cycle from tn_1 to $tn_{k'}$ satisfies the definition of an empty cycle (Def. 12). □

The A* graph search in the presence of a grow and shrink cycle is incomplete. To illustrate how this breaks the algorithm, consider $T \rightarrow x \mid TT \mid \epsilon$ which only differs from the previous example in that T has an empty method. T clearly has a grow cycle $T \rightarrow TT$, and allows shrinking with $T \rightarrow \epsilon$. We know that each run of the cycle increases the network size (from the grow trait), and the *OPEN* list keeps growing. However, at the same time, a perfect heuristic cannot rule out that the algorithm is not on the correct path because the resulting task networks can freely go back to a prior form with the shrink trait. A few iterations of the search are depicted in Fig. 1.

Theorem 4. *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN problem such that there exists a $c \in C$ where c is the initiator of a grow and shrink cycle and c is f-Reachable under h^*. Graph search A* with h^* is incomplete.*

Proof. Let $SN_1, \ldots, SN_{k+1}$ be the sequence of search nodes corresponding to the grow and shrink cycle initiated from c. By the f-Reachability assumption, we know that SN_1 may eventually be expanded by A*. From Corollary 1, we know that every grow and shrink cycle lies on an empty cycle. From theorem 1, it follows that $h^*(SN_1) = h^*(SN_2) = \cdots = h^*(SN_{k+1})$. Hence, SN_k may be expanded. Its successor SN_{k+1} is not isomorphic to SN_1 as the task network has changed. Thus, it will not be in the *CLOSED* list, and will be inserted to the *OPEN* list. The same process may start again from SN_{k+1}, and the algorithm may or may not terminate based on the tie-breaking strategy. □

In the proof of Thm. 4, the termination is determined by the tie-breaking strategy. However, if we relax the perfect heuristic to almost perfect A* is bound to get stuck in an inescapable cycle, regardless of the tie-breaking strategy.

Theorem 5. *Let h^Δ be any almost perfect heuristic. Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN problem such that there exists a $c \in C$ where c is the initiator of a grow and shrink cycle and c is f-Reachable under h^Δ. Graph search A* with h^Δ does not terminate on P.*

Proof. Let g^* be the optimal cost for problem P. By the f-Reachability assumption we know that there exists a search node $SN = \langle \Gamma, \overline{c\beta}, s \rangle$ where $\sum_{a \in \Gamma} cost(a) + h^\Delta(\overline{c\beta}, s) \leq g^*$. Assuming $g^* \neq \infty$ (i.e., the problem has a solution), we can substitute $h^\Delta(\overline{c\beta}, s)$ with $\max(h^*(\overline{c\beta}, s) - \Delta, 0)$ to obtain $\max(g^* - \Delta, 0) \leq g^*$ which is always true since g^* is non-negative. We know that A* will terminate only after there is no search node SN' in the *OPEN* list such that $f(SN') < g^*$ (Pearl 1984). Hence, the termination condition does not fire. □

Our incompleteness results are summarized in table 1. A counter-intuitive implication of Theorems 2 and 3 can be observed in this table; having a *CLOSED* list has a stronger effect on completeness than having the perfect heuristic! Previous results attributed the increased coverage (i.e., number of solved problems) of the PANDA planning system (Höller et al. 2020) – the framework adopted by the winners of optimal HTN planning tracks of the IPC 2023 – after introducing duplicate detection mechanism (i.e., switching from tree search to graph search) to reduced duplicate computational efforts (Höller and Behnke 2021). However, given the new evidence, we hypothesize that this mechanism allowed

	None	Unrestricted	Empty	Growing ϵ-prefix	Grow & Shrink
Tree search A* with h^*	✓	✓			
Graph search A* with h_0	✓	✓	✓		
Graph search A* with h^Δ	✓	✓	✓	✓	
Graph search A* with h^*	✓	✓	✓	✓	

Table 1: The completeness of A* variants when faced with different cycle types. A checkmark indicates that the algorithm in the corresponding row terminates with an answer even if the cycle in the corresponding column is initiated during the search.

the algorithm to deal with empty cycles. The evaluation section backs up our hypothesis by showing that such cycles are present in many of the domains.

Cycle Existence Test

We know for a fact that empty cycles can only happen if there is a method with zero or one subtask; otherwise, the expansion results in a larger task network (Behnke, Höller, and Biundo-Stephan 2019). Thus, if such a method does not exist (i.e., the problem is in $NF_{\geq 2}$ form (Höller et al. 2014)), the only cycle that can be present is the growing ϵ-prefix one. The existence of any cycle introduced in this paper can be checked using the Task Decomposition Graph (TDG) (Bercher et al. 2017; Elkawkagy et al. 2012).

Definition 16 (Task Decomposition Graph)**.** *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN planning problem. The bipartite graph $\phi = \langle V_T, V_M, E_{T \to M}, E_{M \to T} \rangle$ is the Task Decomposition Graph of P iff:*

1. *$c_I \in V_T$,*
2. *if $t \in V_T$ and there exists a method $m = \langle t, tn \rangle$, then $m \in V_M$ and $\langle t, m \rangle \in E_{T \to M}$,*
3. *if $m = \langle t, \overline{t_1 \dots t_k} \rangle \in V_M$, then $\forall t_i \in \overline{t_1 \dots t_k} \colon t_i \in V_T$ and $\langle m, t_i \rangle \in E_{M \to T}$,*
4. *ϕ is minimal w.r.t. conditions 1 to 3.*

From definition, it follows that the search space of a problem has a cycle if and only if its TDG has a cycle.

Definition 17 (TDG Cycle)**.** *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN planning problem, and $\phi = \langle V_T, V_M, E_{T \to M}, E_{M \to T} \rangle$ be its TDG. P has a cycle iff there exists a sequence of nodes $t_1, m_1, t_2, m_2, \dots m_k, t_k$ such that $t_k = t_1$ and for all $\langle t_i, m_i, t_{i+1} \rangle$ where $1 \leq i < k$, it holds that $\langle t_i, m_i \rangle \in E_{T \to M}$, and $\langle m_i, t_{i+1} \rangle \in E_{M \to T}$.*

Given a cycle in the TDG, we can classify it by checking the introduced prefix and suffix. For that, we need to compute the set of all tasks that can be reduced to epsilon, C_ϵ, which is known to be polynomial (Behnke, Höller, and Biundo 2015; Hopcroft, Motwani, and Ullman 2006).

Proposition 1. *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN planning problem, and $\phi = \langle V_T, V_M, E_{T \to M}, E_{M \to T} \rangle$ be its TDG. Let σ be a cycle in ϕ where:*

$$\sigma = t_1, \langle t_1, \overline{\alpha_1 t_1 \beta_1} \rangle, t_2, \langle t_2, \overline{\alpha_2 t_2 \beta_2} \rangle, \dots, \langle t_n, \overline{\alpha_n t_n \beta_n} \rangle, t_1$$

and $\alpha_i, \beta_i \in T^*$ for all $1 \leq i \leq n$. Let $C_\epsilon \subseteq C$ be the set of compound tasks that can be reduced to ϵ. The following statements are implied:

1. *P has an ϵ-prefix cycle iff for all α_i either we have $\alpha_i = \epsilon$, or $\forall t \in \alpha \colon t \in C_\epsilon$.*
2. *P has an empty cycle iff for all α_i and β_i, either we have $\alpha_i \beta_i = \epsilon$, or $\forall t \in \alpha_i \beta_i \colon t \in C_\epsilon$.*
3. *P has a growing ϵ-prefix cycle iff in addition to the conditions for an ϵ-prefix cycle, we have $\beta_1 \beta_2 \cdots \beta_n \neq \epsilon$.*
4. *P has a grow and shrink cycle iff in addition to the conditions for a growing ϵ-prefix cycle, we have $\forall t \in \beta_1 \beta_2 \cdots \beta_n \colon t \in C_\epsilon$.*

A topological sorting of the TDG establishes whether a cycle exists or not in polynomial time (Russell and Norvig 2010). However, a naive approach to cycle characterization requires enumeration of all cycles in the TDG, which is exponential with respect to the number of nodes (Johnson 1975). Thus, it remains to be seen whether an efficient algorithm for classification of the cycles exist or not. However, we provide an algorithm to check for ϵ-prefix cycles in polynomial time. This algorithm cannot differentiate between the three possible ϵ-prefix cycles (empty, growing, grow and shrink). The underlying idea is to compute the set of symbols that can appear in the first position of a decomposed task network, and the implementation (Alg. 2) is a slightly modified version of the "first" set in compiler design (Aho et al. 2006). If a compound task appears in its first set, it follows that it can do recursion without introducing any other symbols (i.e., an ϵ-prefix cycle). In the worst case, the algorithm needs to compute as many as $|A \cup C|$ first sets. Since, each set can contain at most the entire set of tasks, the fixed-point iteration terminates in a low-order polynomial time with respect to the number of tasks.

Algorithm 2: ϵ-Prefix Cycle Existence Test

Input: TOHTN Planning Problem $P = \langle F, C, A, M, c_I, s_I \rangle$, and a set of nullable compound tasks C_ϵ.
Output: True, or False

```
foreach a ∈ A do first(a) = {a} ;
foreach c ∈ C do first(c) = ∅ ;
changed ← True;
while changed = True do
    changed ← False;
    foreach ⟨c, t_1 t_2 ... t_k⟩ ∈ M do
        before ← first(c);
        i ← 1;
        for i ≤ k do
            first(c) ← first(c) ∪ {t_i} ∪ first(t_i);
            if t_i ∉ C_ε then break;
        if before ≠ first(c) then changed ← True;
foreach c ∈ C do
    if c ∈ first(c) then return True;
return False
```

The question of whether a cycle is triggered during search or not, relies on the f-Reachability of the initiator (Def. 13). Thus, a tight decision procedure to check whether A* terminates on a particular TOHTN problem is not practically feasible. The following theorem establishes that f-Reachability is as hard as plan existence.

Theorem 6. *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN problem, and $c \in C$ be a compound task. Deciding whether c is f-Reachable with h^* is EXPTIME-hard.*

Proof. We reduce from the TOHTN plan existence problem, which is known to be EXPTIME-hard (Alford, Bercher, and Aha 2015). Given an arbitrary TOHTN problem P,

we construct a problem $P' = \langle F, C', A, M', c'_I, s_I \rangle$ where $C' = C \cup \{c_f, c'_I\}$ such that $C \cap \{c_f, c'_I\} = \emptyset$, and $M' = M \cup \{\langle c'_I, \overline{c_I c_f} \rangle, \langle c_f, \epsilon \rangle\}$. In other words, we create a planning problem with a newly created compound task, which can only be decomposed to ϵ, appended as a suffix to the original problem. The construction is clearly poly-time. Notice that the optimal solution cost of P' is equal to P because c_f can be removed with a cost of zero. Next, we prove that c_f is f-Reachable in P' if and only if P has a solution.

$\Rightarrow$ If P has a solution, by definition, there exists a sequence of progressions $\overline{c_I} \overset{p_1}{\rightsquigarrow} tn_2 \overset{p_2}{\rightsquigarrow} \dots \overset{p_m}{\rightsquigarrow} \epsilon$ with an optimal solution cost of g^*. By construction, applying the same progression to $\overline{c_I c_f}$ results in $tn = \overline{c_f}$ in which tn satisfies the first condition for f-Reachability. Since the sequence of progressions leads to a solution, the second condition is also satisfied. Given that c_f can only be decomposed to an empty task network and the fact that h^* is admissible, it follows that $h^*(tn, s) = 0$ for all $s \in 2^F$. Thus, the f-Reachability equation simplifies to $\sum_{i=1}^{m} cost(p_i) \leq g^*$. With further simplification, we obtain $g^* \leq g^*$ which trivially holds. In conclusion, if P has a solution, c_f is f-Reachable in P'.

$\Leftarrow$ if c_f is f-Reachable, by definition there exists a sequence of progressions $\overline{c'_I} \overset{p_1}{\rightsquigarrow} tn_2 \overset{p_2}{\rightsquigarrow} \dots \overset{p_m}{\rightsquigarrow} \overline{c_f \beta}$ with $\beta \in T^*$. As c_f does not appear in the original problem, we know that the only f-Reachable progression sequence is one that leads to $\overline{c_f}$ with no suffix. Since c_f was added as a last task, and progression processes a suffix if and only if its prefix has been progressed away (Höller et al. 2020; Alford et al. 2012), it follows that c_I is refined by this sequence. The second condition of f-Reachability guarantees that this refinement is a solution. □

Handling Cycles

The complete elimination of cycles from a domain is neither feasible nor desirable, as it would result in an overly restrictive class of problems. Such an acyclic formulation, while potentially offering reduced computational complexity (Alford, Bercher, and Aha 2015), would significantly limit the expressive power of HTN planning (Höller et al. 2016). The first strategy is to compute decomposition bounds on known problems, and restrict recursion depth to that height (Alford et al. 2016). For an arbitrary TOHTN planning problem, this bound is known to be $|C| \times (2^{|F|})^2$ (Behnke, Höller, and Biundo 2018). While the bound is far from ideal, it makes the search space finite. Thus, ensuring the completeness of A*. The second strategy is to focus on the cost of going through a cycle. The ϵ-prefix cycle demands that no primitive action needs to be executed to do recursion (Def. 11). In other words, the only way of incurring a cost is eliminated. The definitions of empty cycle (Def. 12), growing ϵ-prefix cycle (Def. 14), and grow & shrink cycle (Def. 15) all rely on the existence of an ϵ-prefix cycle. Thus, eliminating the ϵ-prefix aspect of a cycle leads to the removal of all problematic cycles. This can be achieved by transforming the grammatical structure of a TOHTN problem to the Greibach Normal Form (GNF) (Greibach 1965). This ensures that a primitive task is executed before every recursion.

Theorem 7. *Let $P = \langle F, C, A, M, c_I, s_I \rangle$ be a TOHTN planning problem with an ϵ-prefix cycle. There exists a transformation to problem $P' = \langle F, C', A, M', c'_I, s_I \rangle$ where there is no ϵ-prefix cycle and the sets of solutions for P and P' are identical.*

Proof. Let $G = \langle C, A, M, c_I \rangle$ be a context-free grammar where C is a set of non-terminal symbols, A is a set of terminal symbols ($A \cap C = \emptyset$), M is a set of production rules, and $c_I \in C$ is the initial symbol. From the GNF transformation (Greibach 1965), we know that there exists a grammar $G' = \langle C', A, M', c'_I \rangle$ such that it produces the same language as G, and all production rules (except for symbol c'_I which can go directly to ϵ) are of the form $Z \to aY_1 \dots Y_k$ where $a \in A$ and $\forall Y_i \in Y_1 \dots Y_k \colon Y_i \in C'$. Thus, we can construct $P' = \langle F, C', A, M', c'_I, s_I \rangle$ where there is no ϵ-prefix cycle (even though it incurs an exponential blow-up because of ϵ rule eliminations (Hopcroft, Motwani, and Ullman 2006)). Next, we need to prove that this transformation preserves the set of solutions. It is known that the set of solutions for P can be defined as $Sol(P) = \mathcal{L}(G) \cap Exec(P)$ where $\mathcal{L}(G)$ is the set of words produced from grammar G (i.e., ignoring executability) and $Exec(P)$ is the set of all executable sequence of primitive tasks (i.e., ignoring hierarchical reachability) (Höller et al. 2014). Since F, A, and s_I, which are concerned with state-reachability are unaltered, it follows that all sequences of primitive tasks that are executable in P are also executable in P'. Hence, $Exec(P) = Exec(P')$. From this, and the language-preservation properties of the Greibach transformation, we can establish that $Sol(P) = Sol(P')$. □

Evaluation

To assess the practical relevance of our theoretical findings, we analyzed all 23 totally-ordered domains from the hierarchical track of IPC 2023[2]. We have observed that not only at least one form of cycle is present almost every domain, but also in many cases there are multiple initiators. Our analysis enumerated all cycles in the TDG of the domains in their lifted representation and ignored the f-reachability of the initiators, which is problem-dependent. It is important to note that the Monroe problems (partially, and fully observable) do not have a unique domain as each problem instance has its own domain. We only considered the domain associated with the first problem in the benchmark set (i.e., `pfile01.hddl`). The number of cycle initiators, as illustrated in Tab. 2, suggest that our investigation is not merely a theoretical construct. To summarize, the table shows that:

1. Cycles are ubiquitous in HTN planning with only two domains not having any form of recursion.
2. Many domains (12 out of 23) have some form of zero-cost recursion. 5 of these domains contain an empty cycle initiator, which means that any failure with tree search A* could be attributed to incompleteness. 9 domains feature a growing ϵ-prefix cycle initiator, which not only necessities a graph search to make A* complete, but also an

[2]The domains are publicly available at:
`www.github.com/ipc2023-htn/ipc2023-domains`

Domain	Compound Tasks			Number of Cycle Initiators Per Type				
	$\|M\|$	$\|C\|$	$\|C_\epsilon\|$	Unrestricted	ϵ-prefix	Empty	Growing ϵ-prefix	Grow & Shrink
AssemblyHierarchical	17	4	0	2	1	**1**	0	0
Barman-BDI	22	10	9	**0**	0	0	0	0
Blocksworld-GTOHP	8	4	0	1	1	0	**1**	0
Blocksworld-HPDDL	12	5	1	**2**	0	0	0	0
Depots	12	6	0	1	1	0	**1**	0
Factories-simple	10	5	3	4	3	0	**3**	0
Freecell-Learned-ECAI-16	245	82	16	50	32	0	**32**	0
Hiking	15	8	0	3	1	0	**1**	0
Lamps	15	6	5	**5**	0	0	0	0
Logistics-Learned-ECAI-16	42	14	5	5	2	0	**2**	0
Minecraft-Player	19	8	7	5	4	4	1	**1**
Minecraft-Regular	14	7	7	3	3	3	1	**1**
Monroe-Fully-Observable	61	39	0	**5**	0	0	0	0
Monroe-Partially-Observable	69	43	0	**5**	0	0	0	0
Multiarm-Blocksworld	12	5	1	**2**	0	0	0	0
Robot	11	6	1	**2**	0	0	0	0
Rover-GTOHP	16	10	0	**1**	0	0	0	0
Satellite-GTOHP	10	6	0	**3**	0	0	0	0
SharpSAT	34	13	9	5	4	**4**	0	0
Snake	5	2	2	**2**	0	0	0	0
Towers	8	5	1	3	1	**1**	0	0
Transport	6	4	0	1	1	0	**1**	0
Woodworking	19	6	0	**0**	0	0	0	0

Table 2: The table presents 23 planning domains present in the hierarchical track of IPC 2023, showing the number of methods ($|M|$), compound tasks ($|C|$), and nullable compound tasks ($|C_\epsilon|$) for each domain. It also quantifies five different types of cycle initiators across domains, with the most dominant cycle initiator type for each domain highlighted in bold. In the presence of an empty cycle, a graph search is essential for completeness, regardless of the heuristic function. On the other hand, in the presence of a growing ϵ-prefix cycle, having a *CLOSED* list is not sufficient to establish completeness. Graph search A*, even with the perfect heuristic, remains incomplete in the face of a grow and shrink cycle.

informed heuristic. Grow and shrink cycles are present in 2 domains. Hence, graph search A* with perfect heuristic may not solve every IPC problem.

Upon further analysis, it is apparent that in domains like Transport, the growing ϵ-prefix cycle emerges naturally from the path-finding behavior. In other words, to go from location A to B, a vehicle first needs go to an intermediate location C, leading to an immediate left-recursive decomposition pattern. However, in Freecell-Learned-ECAI-16, the same cycle arises from a long chain of decompositions. The grow and shrink cycle in both of the Minecraft domains suggest that such cycles also arise naturally in practice when trying to simulate "loop until a condition is met". This is the case for the compound task `buildrow` in the Minecraft domains where we want to create n rows in a certain direction. The cycle is first initiated by a recursive call to build a single row in a particular location, which creates the ϵ-prefix part. The second (and the last) subtask in the method calls another compound task to figure out how to place a block in the adjacent location. However, if such a block is already in place, an empty method can be applied to remove the task; hence, forming a grow and shrink cycle. We conclude that cycle types are an important structural consideration before attempting to solve the problem at hand. Our experimental setup is publicly available (Yousefi et al. 2025).

Conclusion

This study uncovers inherent limitations in the application of A* to HTN planning. The kind of cycles that zero-cost recursion can create is powerful enough to render even the perfect heuristic insufficient to establish completeness. Interestingly, the totally-ordered case of HTN planning that we studied in this paper are known to be decidable. Our evaluation suggests that many of the cycles defined in this paper happen frequently in real-world scenarios. However, we provide a polynomial-time test to detect the problematic decomposition patterns, and a mapping that preserves domain semantics while ensuring the completeness of A* in the transformed space.

Acknowledgements

Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

Aghighi, M.; and Backstrom, C. 2016. A Multi-Parameter Complexity Analysis of Cost-Optimal and Net-Benefit Planning. In *Proc. of the 26th ICAPS*, 2–10. AAAI Press.

Aho, A. V.; Lam, M. S.; Sethi, R.; and Ullman, J. D. 2006. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., 2nd edition.

Alford, R.; Behnke, G.; Höller, D.; Bercher, P.; Biundo-Stephan, S.; and Aha, D. W. 2016. Bound to Plan: Exploiting Classical Heuristics via Automatic Translations of Tail-Recursive HTN Problems. In *Proc. of the 26th ICAPS*, 20–28. AAAI Press.

Alford, R.; Bercher, P.; and Aha, D. W. 2015. Tight Bounds for HTN Planning. In *Proc. of the 25th ICAPS*, 7–15. AAAI Press.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2012. HTN Problem Spaces: Structure, Algorithms, Termination. In *Proc. of the 5th SoCS*, 2–9. AAAI Press.

Behnke, G.; Höller, D.; and Biundo-Stephan, S. 2019. Finding Optimal Solutions in HTN Planning – A SAT-based Approach. In *Proccedings of 28th IJCAI*, 5500–5508. IJCAI.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the Complexity of HTN Plan Verification and Its Implications for Plan Recognition. In *Proc. of the 25th ICAPS*, 25–33. AAAI Press.

Behnke, G.; Höller, D.; and Biundo, S. 2018. totSAT – Totally-Ordered Hierarchical Planning Through SAT. In *Proc. of the 32nd AAAI*, 6110–6118. AAAI Press.

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmüller, R.; and Kambhampati, S. 2010. G-value plateaus: A Challenge for Planning. In *Proc. of the 20th ICAPS*, 259–262. AAAI Press.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *Proc. of the 28th IJCAI*, 6267–6275. IJCAI.

Bercher, P.; Behnke, G.; Höller, D.; and Biundo, S. 2017. An Admissible HTN Planning Heuristic. In *Proc. of the 26th IJCAI*, 480–488. IJCAI.

Edelkamp, S.; Schroedl, S.; and Koenig, S. 2010. *Heuristic Search: Theory and Applications*. Morgan Kaufmann Publishers Inc.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving Hierarchical Planning Performance by the Use of Landmarks. In *Proc. of the 26th AAAI*, 1763–1769. AAAI Press.

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

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *Proc. of the 22nd IJCAI*, 1955–1961. AAAI Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Greibach, S. A. 1965. A New Normal-Form Theorem for Context-Free Phrase Structure Grammars. *JACM*, 12(1): 42–52.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Helmert, M.; and Röger, G. 2008. How good is almost perfect? In *Proc. of the 23rd AAAI*, 944–949. AAAI Press.

Höller, D. 2023. The PANDA Progression System for HTN Planning in the 2023 IPC. In *Proc. of the 11th IPC: Planner Abstracts – HTN Planning Track*.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *Proc. of the 21st ECAI*, 447–452. IOS Press.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2006. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Longman Publishing Co., Inc.

Höller, D.; and Behnke, G. 2021. Loop Detection in the PANDA Planning System. In *Proc. of the 31st ICAPS*, 168–173. AAAI Press.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages. In *Proc. of the 26th ICAPS*, 158–165. AAAI Press.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020. HTN Planning as Heuristic Progression Search. *JAIR*, 67: 835–880.

Johnson, D. B. 1975. Finding All the Elementary Circuits of a Directed Graph. *SIAM Journal on Computing*, 4(1): 77–84.

Korf, R. E. 1985. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. *AIJ*, 27(1): 97–109.

Lin, S.; and Bercher, P. 2022. On the Expressive Power of Planning Formalisms in Conjunction with LTL. In *Proc. of the 32nd ICAPS*, 231–240. AAAI Press.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *JAIR*, 20: 379–404.

Nilsson, N. J. 1982. *Principles of artificial intelligence*. Springer Science & Business Media.

Olz, C.; and Bercher, P. 2023. A Look-Ahead Technique for Search-Based HTN Planning: Reducing the Branching Factor by Identifying Inevitable Task Refinements. In *Proc. of the 16th SoCS*, 65–73. AAAI Press.

Olz, C.; Höller, D.; and Bercher, P. 2023. The PANDADealer System for Totally Ordered HTN Planning in the 2023 IPC. In *Proc. of the 11th IPC: Planner Abstracts – HTN Planning Track*.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. USA: Addison-Wesley Longman Publishing Co., Inc.

Russell, S. J.; and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition.

Yousefi, M.; Schmautz, M.; Haslum, P.; and Bercher, P. 2025. Experimental Setup for the ICAPS 2025 paper: "How Good is Perfect? On the Incompleteness of A* for Total-Order". Available at: http://doi.org/10.5281/zenodo.15320482.