

Flexible FOND HTN Planning: A Complexity Analysis

Dillon Z. Chen, Pascal Bercher

The Australian National University, Canberra, Australia
 {dillon.chen, pascal.bercher}@anu.edu.au

Abstract

Hierarchical Task Network (HTN) planning is an expressive planning formalism that has often been advocated to address real-world problems. Yet few extensions exist that can deal with the many challenges encountered in the real world, one being the capability to express uncertainty. Recently, a new HTN formalism for fully observable nondeterministic problems was proposed and studied theoretically. In this paper, we lay out limitations of that formalism and propose an alternative definition, which addresses and resolves such limitations. We also study its complexity for certain problems.

Introduction

Hierarchical task network (HTN) planning is a planning approach that focuses on problem decomposition. Compound tasks describe abstract activities, and the domain model describes how they can be carried out by exploiting decomposition methods, pre-defined recipes stating by which plans such compounds tasks may be implemented.

The goal is to find a plan – a sequence of primitive tasks that can be executed – which successfully implements the given initial compound tasks defining the planning problem. Because this task hierarchy may be exploited to encode expert knowledge and thus gives another means of modelling a problem, and because it may be used to also *exclude* undesired solutions, it has been used in many different practical scenarios (Bercher, Alford, and Höller 2019).

In particular when facing real-world problems, we may face challenges and limitations if we model deterministically. The world may be dynamically changing (Patra et al. 2020, 2021; Li, Patra, and Nau 2021), partially observable (Richter and Biundo 2017), or require reasoning over actions with nondeterministic outcomes (Kuter and Nau 2004; Kuter et al. 2005, 2009) – most of these works in the realm of HTN planning and uncertainty focused on developing planners that produce classical policies to (non-hierarchical) fully observable nondeterministic (FOND) problems.

Recently, we proposed such an extension of FOND policies capable of capturing solutions to FOND HTN problems where solutions need to be refinements of an initial task network (Chen and Bercher 2021), just like in standard

deterministic HTN planning. For this FOND HTN formalisation, we studied the computational complexity of various restrictions on the task hierarchy, as well as the impact of two ways uncertainty is taken into account: during planning time (linearisation-dependent solutions), or during execution time (outcome-dependent solutions). The latter more flexible solution definition states that a primitive partially ordered plan is regarded a solution when after the execution of any nondeterministic action one is still able to continue executing the plan by picking an appropriate action depending on previous action outcomes until all actions of the plan are successfully executed. This definition assumes that one still needs to compute one such policy before one starts executing it. Thus we will denote this formalism as FOND^{FM} HTN, indicating that they use “fixed methods” in their solutions.

In this paper we propose a more flexible FOND HTN formalisation where policies are no longer defined based on primitive plans, but allow a selection of decomposition methods for compound tasks in the policy, which therefore allows this choice to depend on the outcome of executed tasks. We thus denote the novel formalism FOND^{MP} HTN, indicating that it uses “method-based policies”. Later (cf. Fig. 1) we show that the FOND^{MP} HTN formalism is indeed more expressive as it allows to find solutions that cannot be found with the FOND^{FM} HTN formalism. On top of the more flexible formalism we present a complexity study for the plan existence problem alongside baseline search algorithms for such problems.

Formalism

The definitions of the FOND^{MP} HTN planning domain are the same of that for FOND^{FM} HTN planning by extending definitions for deterministic HTN planning (Geier and Bercher 2011; Bercher, Alford, and Höller 2019) to include nondeterministic actions.

Definition 1. A *task network* tn is a tuple $\langle T, \prec, \alpha \rangle$ where T is a finite set of *task id symbols* or *labels*, $\prec \subseteq T \times T$ is a strict partial order on T , and $\alpha : T \rightarrow \mathcal{N}$ maps a task id to some task name in the set of task names $\mathcal{N}$.

We also have notation for special task networks: $tn(a) = \langle \{t\}, \emptyset, \{(t, a)\} \rangle$ and $tn_\emptyset = \langle \emptyset, \emptyset, \emptyset \rangle$. The notation $|_{T'}$ applied to a task network or any of its elements denotes restriction on the respective sets to only tasks in T' . Also let

Hierarchy	Order	Weak		Strong		Strong cyclic	
primitive	total partial	NP NP	Prop. 2 Prop. 2		P* PSPACE		Cor. 1 Cor. 1
acyclic	total partial	PSPACE NEXPTIME	Prop. 2 Prop. 2	EXPTIME EXPSPACE	Thm. 1 Thm. 2	EXPTIME EXPSPACE	Thm. 1 Thm. 2
regular	total partial	PSPACE PSPACE	Prop. 2 Prop. 2	EXPTIME EXPTIME	Thm. 3 Thm. 3	EXPTIME EXPTIME	Thm. 3 Thm. 3
tail-recursive	total partial	PSPACE EXPSPACE	Prop. 2 Prop. 2	EXPTIME 2-EXPTIME	Thm. 4 Thm. 5	EXPTIME 2-EXPTIME*	Thm. 4 Thm. 5

Table 1: Complexity results for FOND^{MP} HTN planning. The first column lists known special cases by restricting the hierarchy, where the general case is undecidable. Classes marked * are not complete where only membership is known.

$tn \setminus t$ denote $tn|_{T \setminus \{t\}}$ with T being the task ids of tn .

We define an equivalence between two task networks which might have the same underlying structure but different task id symbols. Specifically, we say that two task networks $tn = \langle T, \prec, \alpha \rangle$ and $tn' = \langle T', \prec', \alpha' \rangle$ are *isomorphic* if there exists a bijection $\sigma : T \rightarrow T'$ between task id symbols where for all $t_1, t_2 \in T$, we have $(t_1, t_2) \in \prec$ iff $(\sigma(t_1), \sigma(t_2)) \in \prec'$ and $\alpha(t) = \alpha'(\sigma(t))$ for all $t \in T$. This definition of equivalence will be required for building well defined FOND^{MP} HTN problems and solutions.

Definition 2. A FOND^{MP} HTN domain $\mathcal{D}$ is a tuple $\langle \mathcal{F}, \mathcal{N}_P, \mathcal{N}_C, \delta, \mathcal{M} \rangle$ where $\mathcal{F}$ is a finite set of *facts*, $\mathcal{N}_P$ is a finite set of *primitive task names*, $\mathcal{N}_C$ is a finite set of *compound task names*, $\delta : \mathcal{N}_P \rightarrow \mathcal{A}$ is an action mapping, and $\mathcal{M}$ is a finite set of *decomposition methods*, with $\mathcal{N}_P \cup \mathcal{N}_C = \mathcal{N}$ disjoint and $\mathcal{A} \subseteq 2^{\mathcal{F}} \times 2^{2^{\mathcal{F}} \times 2^{\mathcal{F}}}$ denoting the set of nondeterministic primitive tasks. A *primitive task* or *action* is defined by a single precondition, and a set of mutually-exclusive alternative possible effects. More specifically, $a = (\text{pre}(a), \text{eff}(a))$ with $\text{eff}(a) = \{(\text{add}_i(a), \text{del}_i(a)) \mid 1 \leq i \leq k\}$ for k dependent on a and $\text{pre}(a), \text{add}_i(a), \text{del}_i(a) \subseteq \mathcal{F}$. For ease of notation whenever we have a deterministic action ($|\text{eff}(a)| = 1$) we will use $(\text{pre}(a), \text{add}(a), \text{del}(a))$ as to remove redundant brackets.

Define a set of states $S = 2^{\mathcal{F}}$ corresponding to subsets of $\mathcal{F}$. Let $\tau : \mathcal{A} \times S \rightarrow \{\top, \perp\}$ denote *executability* of an action at a state where $\tau(a, s) = \top$ for $\text{pre}(a) \subseteq s$ and $\tau(a, s) = \perp$ otherwise. For ease of notation, we also define the executability function τ for primitive task names and id symbols in the obvious way by $\tau(n, s) = \tau(\delta(n), s)$ and $\tau(t, s) = \tau(\alpha(t), s) = \tau(\delta(\alpha(t)), s)$ for $n \in \mathcal{N}_P$ and $t \in T$ respectively. We also define an application function $\gamma : \mathcal{A} \times S \rightarrow 2^S$ where for $a \in \mathcal{A}, s \in S$ we have $\gamma(a, s)$ undefined if $\tau(a, s) = \perp$ and otherwise we have $\gamma(a, s) = \{(s \setminus \text{del}_i(a)) \cup \text{add}_i(a) \mid 1 \leq i \leq |\text{eff}(a)|\}$.

Now that we have the distinction of primitive and compound task names ($\mathcal{N}_P$ and $\mathcal{N}_C$, respectively), we can define two more concepts: Define γ on primitive task names $n \in \mathcal{N}_P$ and id symbols by $\gamma(n, s) = \gamma(\delta(n), s)$ and $\gamma(t, s) = \gamma(\alpha(t), s) = \gamma(\delta(\alpha(t)), s)$. We can now say that $tn = \langle T, \prec, \alpha \rangle$ is a *primitive task network* if all its tasks are primitive, i.e., for all $t \in T$, we have $\alpha(t) \in \mathcal{N}_P$.

Definition 3. A (*decomposition*) *method* is a tuple $m =$

(c, tn_m) with $c \in \mathcal{N}_C$ and $\text{tn}_m = \langle T_m, \prec_m, \alpha_m \rangle$. We can apply m to $\text{tn}_1 = \langle T_1, \prec_1, \alpha_1 \rangle$ if there exists $t \in T_1$ with $\alpha_1(t) = c$. We say m decomposes t in tn_1 to generate a task network $\text{tn}_2 = \langle T_2, \prec_2, \alpha_2 \rangle$ with $T_2 := T_1' \cup T_m'$, $\alpha_2 := (\alpha_1 \cup \alpha_m')|_{T_1'}$ and $\prec_2 := (\prec_1 \cup \prec_m')|_{T_1' \cup \{(t_1, t_2) \in T_1' \times T_m' \mid (t_1, t_2) \in \prec_1\} \cup \{(t_1, t_2) \in T_m' \times T_1' \mid (t, t_2) \in \prec_1\}}$, where $T_1' = T_1 \setminus \{t\}$ and tn_m' is a task network isomorphic to tn_m such that $T_1' \cap T_m' = \emptyset$. We denote this application by $\text{tn}_1 \rightarrow_m^t \text{tn}_2$.

Definition 4. A FOND^{MP} HTN problem $\mathcal{P}$ is a tuple $\langle \mathcal{D}, s_I, \text{tn}_I \rangle$ with $\mathcal{D}$ a FOND^{MP} HTN domain, $s_I \in 2^{\mathcal{F}}$ an initial state and tn_I an initial task network.

With a FOND^{MP} HTN problem in hand, we now provide explicit definitions for what a plan or solution means. Similar to FOND^{FM} HTN planning, we employ policies to define a solution. The difference between the two formalisms lies in how decomposition plays into a solution. In FOND^{FM} HTN planning, solutions are defined by fixing a sequence of decomposition methods to apply on the initial task network and then constructing a policy for the acquired primitive task network. In contrast, we will integrate methods into our policy, meaning that methods may be applied at different times depending on nondeterministic task effects.

Although FOND^{MP} HTN solutions are more flexible by integrating decomposition into online execution, it has a few drawbacks: policies can grow arbitrarily large and online execution is hard. This arises from how we define a policy to take as input a task network and state, where the set of task networks is possibly unbounded in contrast to outcome-dependent solutions of FOND^{FM} HTN problems which define policies for primitive task networks only using a lookup table of a current state and previously executed tasks. The latter is always bounded by noticing that there are only a finite number of states and subsets of tasks to account for.

Definition 5. Let $\mathcal{D}$ be a FOND^{MP} HTN domain. A *policy* π is a *partial* function $\pi : TN \times S \rightarrow T \times \mathcal{M}'$ where TN is the set of all possible task networks, T is the union of the sets of (corresponding) tasks in the task networks of TN and $\mathcal{M}' = \mathcal{M} \cup \{\varepsilon\}$, where ε is a symbol used for primitive tasks. Specifically, $\langle (tn, s), (t, m) \rangle \in \pi$ for $tn = \langle T, \prec, \alpha \rangle$ only if $t \in T$, and if t is primitive, $m = \varepsilon$, and otherwise if t is compound, $m = (\alpha(t), \text{tn}')$ is a method of $\mathcal{D}$. For brevity's sake, we will refer to task network state tuples

Algorithm 1: Policy Execution Procedure

```

1  $(tn, s) \leftarrow (tn_I, s_I)$ ;
2 while InstructionExists( $\pi, tn, s$ ) do
3    $(t, m) \leftarrow \text{GetInstruction}(\pi, tn, s)$ ;
4   if  $m = \varepsilon$  then
5      $tn \leftarrow \text{Remove}(tn, t)$ ;
6      $\text{Execute}(t)$ ;
7      $s \leftarrow \text{SenseCurrentState}()$ ;
8   else  $tn \leftarrow \text{Decompose}(tn, t, m)$ ;

```

(tn, s) as *subproblems*, given that each (tn, s) can be viewed as a standalone problem to be solved by itself.

We also impose the condition on a policy that for all pairs $\langle (tn_1, s_1), (tn_2, s_2) \rangle \in \pi$, if $s_1 = s_2$, then tn_1 and tn_2 are not isomorphic. This condition is required to create a well defined notion of policy execution.

Execution of a policy for FOND STRIPS planning is described as a reactive execution loop that executes actions based on a survey of the state of the world, which shall also be made explicit for FOND^{MP} HTN planning for a given task network tn_I and state s_I in Algorithm 1.

The function $\text{InstructionExists}(\pi, tn, s)$ returns true if tn is not the empty task network and there exists a task network tn' that is isomorphic to tn such that $\pi(tn', s)$ exists. $\text{GetInstruction}(\pi, tn, s)$ returns $\pi(tn', s)$, assuming that $\text{InstructionExists}(\pi, tn, s)$ is true. $\text{Remove}(tn, t)$ returns the task network tn without task t and $\text{Decompose}(tn, t, m)$ the task network we get when m decomposes t in tn . Lastly, $\text{SenseCurrentState}()$ returns the state of the world.

Having defined a mechanism to execute FOND task networks, we can now describe and formalise FOND^{MP} HTN solution criteria. We define weak, strong and strong cyclic solutions as is canonical to non-hierarchical nondeterministic planning (Cimatti et al. 2003) and also in YoYo, a planner which integrates HTN planning for solving such planning problems (Kuter et al. 2005, 2009). To formalise these concepts we will define the execution structure of a policy as a graph and use this graph structure to define solutions.

Definition 6. Let $\mathcal{P}$ be a FOND^{MP} HTN problem. Let the tuple $L = \langle \mathcal{U}, \mathcal{V} \rangle$ where $\mathcal{U} \subseteq TN \times \mathcal{S}$ and $\mathcal{V} \subseteq (TN \times \mathcal{S}) \times (T \times (\mathcal{M} \cup \{\varepsilon\})) \times (TN \times \mathcal{S})$ are minimal sets satisfying the conditions $(tn_I, s_I) \in \mathcal{U}$, and if $(tn, s) \in \mathcal{U}$ and $\pi(tn, s) = (t, m)$ then if t is primitive, for all $s' \in \gamma(t, s)$ we have $(tn \setminus t, s') \in \mathcal{U}$ and $((tn, s), (t, m), (tn \setminus t, s')) \in \mathcal{V}$, and if t is compound, we have $(tn', s) \in \mathcal{U}$ and $((tn, s), (t, m), (tn', s)) \in \mathcal{V}$ where $tn \rightarrow_m^t tn'$.

The *execution structure* induced by a policy π is the tuple $[L] = \langle [\mathcal{U}], [\mathcal{V}] \rangle$ where $[\mathcal{U}]$ is the set $\mathcal{U}$ quotient subproblems out by the relation $(tn, s) \sim (tn', s)$ iff tn and tn' are isomorphic and similarly for $[\mathcal{V}]$ where $((tn_1, s), (t, m), (tn_2, s)) \sim ((tn'_1, s), (t, m), (tn'_2, s))$ iff tn_1 and tn_2 are isomorphic and tn'_1 and tn'_2 are isomorphic.

For ease of notation, we will omit the equivalence relation notation (i.e. the square brackets) for an execution structure. We can also view an execution structure L as a directed

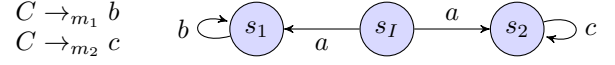


Figure 1: Transition system and domain for which only a FOND^{MP} HTN policy exists given $tn_I = \langle a, C \rangle$.

graph with nodes represented by elements in $\mathcal{U}$ and directed edges by elements in $\mathcal{V}$. Define (tn_I, s_I) to be an *initial node* and any $(tn, s) \in TN \times \mathcal{S}$ to be a *terminal node* if it has no outgoing edges, and a *goal node* if $tn = tn_\emptyset$. We now proceed to define the three solution criteria.

Definition 7. Let $\mathcal{P}$ be a FOND^{MP} HTN problem and tn a task network. Let π be a policy with execution structure $L = \langle \mathcal{U}, \mathcal{V} \rangle$. We say that π is a *weak solution* if L is finite and there exists a terminal node of L that is a goal node, a *strong cyclic solution* if every terminal node of L is a goal node, a *strong (acyclic) solution* if L is finite and acyclic and every terminal node of L is a goal node.

Another way of interpreting the solution criteria is looking at how Algorithm 1 terminates: weak solutions sometimes terminate and if they do, they might do so with an empty task network (but could also fail), strong solutions always terminate with an empty task network (hence the requirement for acyclic L), and strong cyclic solutions eventually terminate (even when L is infinite).

Practically, strong solutions are the most reliable as they guarantee the goal condition be met in finitely many steps. This is followed by strong cyclic solutions which also guarantee that eventually we reach the goal condition. However, execution can be arbitrarily long. Thus, strong solutions are a special case of strong cyclic solutions which are in turn a special case of weak solutions.

Consider the following example (Fig. 1) that proves that our novel FOND^{MP} HTN policy definition that incorporates methods is more expressive than the FOND^{FM} HTN policy. Assume a totally ordered initial task network tn_I containing a primitive task a followed by a compound task C . Depending on nondeterminism, a leads into s_1 or s_2 , but only b is executable in s_1 and only c is executable in s_2 , so no fixed choice of methods can non-weakly solve the problem. Delaying method choice until a was performed does solve it.

Problem Classes

Given that standard HTN planning is undecidable (Erol, Hendler, and Nau 1996; Geier and Bercher 2011), studies have been made to find problem subclasses that can be decided. We list the commonly studied subclasses here (Erol, Hendler, and Nau 1996; Alford et al. 2012; Alford, Bercher, and Aha 2015). We will define *stratifications* proposed by Alford et al. (2012) to help define the latter two.

Definition 8. An HTN problem $\mathcal{P}$ is *primitive* if tn_I is primitive. Note that sets $\mathcal{N}_C$ and $\mathcal{M}$ are now irrelevant.

Definition 9. An HTN problem $\mathcal{P}$ is *regular* if for its initial task network $tn_I = \langle T, \prec, \alpha \rangle$ and for all its methods $(c, \langle T, \prec, \alpha \rangle) \in \mathcal{M}$ it holds that there is at most one compound task in T , and if $t \in T$ is compound, it is the *last* task, meaning that for all $t' \in T$ with $t' \neq t$ we have $t' \prec t$.

Algorithm 2: Alternating Strong Progression Search

```
1 Procedure StrongPlanExistence (tn, s,  $\mathcal{M}$ , V):  
2   if tn = tn0 then return true;  
3   if (tn, s) ∈ V then return false;  
4   V ← V ∪ {(tn, s)};  
5   guess a first task t in tn;  
6   if t is a primitive task then  
7     if not τ(t, s) then return false;  
8     tn ← Remove(tn, t);  
9     return for-all s' ∈ γ(t, s)  
       StrongPlanExistence (tn, s',  $\mathcal{M}$ , V);  
10  else  
11    guess a method m in  $\mathcal{M}$ ;  
12    tn ← Decompose(tn, t, m);  
13    return StrongPlanExistence (tn, s,  $\mathcal{M}$ , V);
```

Definition 10. A *stratification* on a set S is a total order $\leq$ on S . An inclusion-maximal subset $C \subseteq S$ is a *stratum* if for all $x, y \in C$ both $x \leq y$ and $y \leq x$ holds.

Definition 11. An HTN problem $\mathcal{P}$ is *acyclic* if no compound task can reach itself via decomposition. More formally, we can define a stratification on $\mathcal{N}_C$ in $\mathcal{P}$ with $c \leq c'$ if there exists a method $(c, \langle T, \prec, \alpha \rangle) \in \mathcal{M}$ and $\alpha(c') \in T$, and for all $c, c' \in \mathcal{N}_C$, if $c \leq c'$, then $c' \not\leq c$.

Definition 12. An HTN problem $\mathcal{P}$ is *tail-recursive* if we can define a stratification on $\mathcal{N}_C$ of $\mathcal{P}$ where for all methods $(c, \langle T, \prec, \alpha \rangle)$ it holds that if there exists a last compound task $t \in T$, then we have $\alpha(t) \leq c$, and for any non-last compound task $t \in T$, we have $\alpha(t) \leq c$ and $c \not\leq \alpha(t)$.

Note by definition that primitive, regular and acyclic problems are all special cases of tail-recursive problems. We also use the same definitions to describe decomposition methods. For example a regular method is a method whose task network has at most one compound task which has to be last.

Search Algorithms

In this section, we describe two algorithms for determining plan existence of a given FOND^{MP} HTN problem. We do so to provide baseline algorithms for applications and to aid with membership proofs for upcoming complexity proofs. Although not necessarily optimal, they are canonical extensions of other baseline algorithms for HTN planning.

Alternating Progression Search

The first algorithm extends progression search which is considered the canonical search algorithm for solving HTN problems (Alford et al. 2012; Höller et al. 2018, 2020) and also employed in efficient HTN planners such as SHOP, SHOP2 and SHOP3 (Nau et al. 1999, 2003, 2005; Goldman and Kuter 2019). We extend the algorithm by introducing ‘universal’ vertices to the graph, similarly to universal states of an alternating Turing machine (ATM) or AND nodes of an AND/OR-tree, to deal with nondeterminism.

Algorithm 2 provides the procedure for determining plan existence. Given a FOND^{MP} HTN problem $\mathcal{P} = \langle \mathcal{D}, s_I, \text{tn}_I \rangle$ with $\mathcal{D} = \langle \mathcal{F}, \mathcal{N}_P, \mathcal{N}_C, \delta, \mathcal{M} \rangle$, the alternating

procedure StrongPlanExist(tn_I, s_I, $\mathcal{M}$, $\emptyset$) determines if a strong solution for $\mathcal{P}$ exists. The meaning of the input variables tn, s, $\mathcal{M}$ are straightforward. The product set V stores previously progressed task networks and visited states in order to detect cycles and deal with them.

The given ATM progression algorithm is an extension of the textbook progression algorithm used for classical HTN planning to our FOND^{MP} HTN setting. Progression is a search algorithm which makes nondeterministic guesses for choosing whether to execute a random first primitive task or to decompose a compound task. By first task of a task network tn, we mean any task that has no predecessors in tn. After doing so, we remove the chosen task from the task network and change the state if the chosen task was primitive. If a solution exists, then choosing the correct progression steps leads us to an empty task network, satisfying the solution criterion. To extend this concept to nondeterministic domains, we use universal states of ATMs to recursively check whether all possible progressed task networks from an executed nondeterministic action contribute to a solution.

Line 2 checks whether we have progressed away the task network and hence have reached an accepting state of the alternating computation tree. Line 3 checks whether we have visited the subproblem (tn, s) before (which requires checking for task network isomorphism) and enters a rejecting state. Line 4 then updates the previous task network-state tuples. Line 5 makes a nondeterministic choice of a task t with no predecessors in tn.

The remainder of the algorithm performs the progression procedure depending on whether t is primitive or compound. If t is primitive, lines 7 to 9 checks whether t is executable at the progressed state s and if so proceeds to remove t from the progressed task network and then recursively calls the function “for all” possibly progressed states as given by $\gamma(t, s)$. The **for-all** statement represents entering a universal state for an ATM encoding. A more high level interpretation is that we return the logical conjunction of the StrongPlanExistence procedure for all possible progressed states. If t is compound, we guess a method for t and expand the task network at t with such method and proceed with the progression algorithm.

Note that this algorithm can be determined by replacing nondeterministic choices with branching as described in in Alg. 1 by Höller et al. (2018). This branching implements an OR node. We have however also to deal with **for-all** statements, which correspond to AND nodes thus implementing an AND/OR search. The optimisation described in Alg. 2 by Höller et al. (2018) for reducing branching from decomposition methods can also be applied here. The reason we do not provide the deterministic version of the algorithm is to emphasise the usual tools (alternation) required to deal with nondeterministic tasks and for complexity analysis later.

Given that general HTN planning is undecidable, it is not necessarily the case that the described algorithm terminates though we will see later that the algorithm terminates for certain problem subclasses. We can also modify the algorithm to allow for strong cyclic solutions by *not* using the set V of visited subproblems but this will not be provided explicitly as it is not used for complexity proofs later.

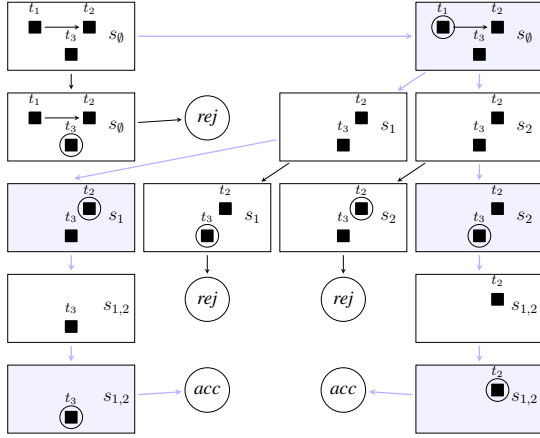


Figure 2: Visualisation of the alternating progression search algorithm. Denote $s_1 = \{1\}$, $s_2 = \{2\}$, $s_{1,2} = \{1, 2\}$.

Fig. 2 provides a visualisation of the high level computation tree associated with the algorithm. We provide the abstract primitive problem it solves as follows. Let $\mathcal{D} = \langle \{1, 2\}, \{a, b, c\}, \emptyset, \delta, \emptyset \rangle$ with δ defined by $a \mapsto (\emptyset, \{(\{1\}, \emptyset), (\{2\}, \emptyset)\})$, $b \mapsto (\{2\}, \{1\}, \emptyset)$, $c \mapsto (\{1\}, \{2\}, \emptyset)$. Then we define the problem by $\mathcal{P} = \langle \mathcal{D}, s_0, \text{tn}_I \rangle$ where $s_0 = \emptyset$ and tn_I is the task network containing primitive tasks a followed by c , unordered to b .

Rectangular nodes in the figure represent search nodes consisting of the currently progressed task network and state. We omit the set V in the visualisation as we do not worry about cycles for primitive problems. Black squares indicate primitive tasks and circles the selection of a task by line 5 of the algorithm. Blue nodes indicate universal states where we have to check that all children nodes are accepting, while the other rectangular nodes indicate existential states. Blue arrows indicate the subtree of the computation tree corresponding to a strong solution.

Bounded Graph Search

In addition to progression search, there is another search technique we can use to determine plan existence if we assume that the number of reachable task networks under progression and state combinations are bounded (e.g. primitive, acyclic, regular and tail-recursive problems). In contrast to progression search or forward search, bounded graph search can be viewed as backwards search with the additional assumption that the search space is bounded. This will be helpful for complexity analysis later as AND/OR search, which incurs additional complexity, is no longer required.

The main idea is that we can generate a bounded search space in the form of a graph for a FOND^{MP} HTN problem. Another way of interpreting this is that we compile a FOND^{MP} HTN problem into a state transition system with initial and goal states and solve the compiled problem similarly to how Cimatti et al. (2003) generates the whole search space as a graph for a non-hierarchical FOND planning problem. Specifically, let $\langle S, A, I, G \rangle$ be a state transition system with S a set of states, $A \subseteq S \times 2^S$ a set of nonde-

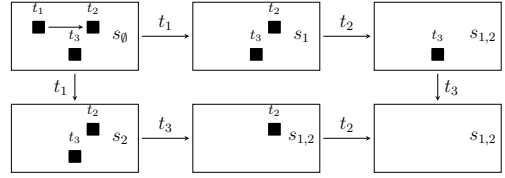


Figure 3: The whole search space of the (primitive) FOND HTN problem from Fig. 2 (where t_3 is not executable in s_0).

terministic actions defined with an action defined as a tuple $(s_\alpha, \{s_1, \dots, s_n\})$ which when applied in s_α can progress to any of the states s_1 to s_n . Next, we have $I \in S$ an initial state and $G \subseteq S$ a set of goal states. The definitions of strong and strong cyclic solutions are similar to that for nondeterministic non-hierarchical planning.

To compile a FOND^{MP} HTN problem into such a system $\langle S, A, I, G \rangle$, we begin by letting S be the set of all possible *reachable task networks* and state tuples. Specifically, the set of reachable task networks TN_R for a problem is defined to be the set of task networks that can be obtained from the initial task network by applying a sequence of first primitive tasks or methods for first compound tasks, quotient out by isomorphism. We will call S the set of *subproblems* given that they can be viewed as HTN problems with the same domain $\mathcal{D}$ and their task networks are part of a solution to the initial task network.

To consider an example, suppose we have a regular HTN problem. Then TN_R includes the initial task network tn_I , the task networks for each method and all task networks that can be reached from tn_I under progression. Since regular problems have at most one compound task in the current task network, TN_R is bounded exponentially corresponding to exponentially many linearisations of these task networks.

In the compilation, we get $I = (\text{tn}_I, s_I)$ and $G = \{(\text{tn}_\emptyset, s) \mid s \subseteq \mathcal{F}\}$ denoting the initial and goal states. Then we define actions by looping through all $\sigma_\alpha = (\text{tn}_\alpha, s_\alpha) \in S$ as follows. For each first task t in tn_α ,

- if t is primitive, define an action (transition)
 $a = (\sigma_\alpha, \{\sigma_i = (\text{tn}_\alpha \setminus \{t\}, s_i) \mid s_i \in \tau(t, s_\alpha)\})$,
- else for each method applicable to t , define an action
 $a = (\sigma_\alpha, \{\sigma_\beta = (\text{tn}_\beta, s_\alpha)\})$ where $\text{tn}_\alpha \rightarrow_m^t \text{tn}_\beta$.

The main idea of such actions is that they connect HTN subproblems depending on if one can reach one subproblem from the other corresponding to an execution of some task.

Fig. 3 illustrates the graph associated with the compiled (primitive) HTN problem described in the section on alternating progression search. Rectangular nodes represent subproblems which are now the states of the compiled classical problem; directed edges represent actions and their effects.

To solve the system viewed as a nondeterministic non-hierarchical planning problem, we employ algorithms for weak, strong and strong cyclic planning in Sections 3 and 4 by Cimatti et al. (2003). All of them run in polynomial time with respect to the size of the graph as they search the graph a number of times to build up a solution.

Now we investigate the complexity of the algorithm by looking at the runtime of the two main steps: building the

graph and solving it. As mentioned in the previous paragraph, solving takes polynomial time with respect to the size of the graph. Building the graph is a bit more involved given that we have to check for graph isomorphism (GI) for equality of nodes. If checking for equality was only constant time, then the time it takes to build the graph is at least exponential in $\mathcal{F}$ as there are exponentially many reachable states, and bounded above polynomially by the size of TN_R . This is because we can build all $2^{|\mathcal{F}|} \cdot |TN_R|$ nodes first then for each node check which other node is reachable from it. Given that there are quadratically many directed edges between nodes, this means that building would take at least exponential time (polynomial with respect to the number of nodes which is exponential), in the order of $2^{|\mathcal{F}|} \cdot |TN_R|$.

Now, if we replace equality checking with GI, we get complexity of order $2^{|\mathcal{F}|} \cdot |TN_R| \cdot f(\max_{tn \in TN_R} |tn|)$ where $f(n)$ denotes the complexity for solving GI for graphs with n vertices. A safe but loose upper bound for f is the exponential function with a brute force algorithm for checking task network isomorphism. In practice, we can apply GI algorithms in literature with upper bound $O(e^{\sqrt{n \log n}})$ (Babai and Luks 1983; Babai, Kantor, and Luks 1983) or recent task network isomorphism solvers (Höller and Behnke 2021). Furthermore, almost all graphs are easy to solve as seen in the `nauty` package (McKay and Piperno 2014).

In our membership proofs of HTN problem classes, the order of $|TN_R|$ will range from polynomial to double exponential with the upper bound on the size of a reachable task network $\max_{tn \in TN_R} |tn|$ ranging from polynomial to exponential. Thus, the complexity we will encounter for this algorithm varies between exponential and double exponential.

Complexity Investigations

This section covers the complexity results for the novel formalism. We show that weak or primitive $FOND^{MP}$ HTN problems are equivalent to weak or primitive $FOND^{FM}$ HTN problems and thus have the same complexities. (I.e., each solution in the one formalism corresponds to a solution in the other.) For strong and strong cyclic problems we have that all acyclic, regular, and tail-recursive classes are made one step harder from their classical counterparts.

We begin by describing how weak and primitive $FOND^{MP}$ HTN and $FOND^{FM}$ HTN planning problems separately have the same semantic definitions which in turn means that the complexity for $FOND^{MP}$ HTN planning is equivalent as that for $FOND^{FM}$ HTN for which there exists a range of results (Chen and Bercher 2021). Specifically, we show that the definitions for weak solutions are equivalent, and also for strong solutions when problems are primitive.

Proposition 1. *The definitions for primitive $FOND^{MP}$ HTN and primitive $FOND^{FM}$ HTN planning are equivalent.*

Proof. First observe a primitive $FOND^{MP}$ HTN policy no longer requires instructions for method applications. Thus, we only need to show that the policies consisting of only primitive task execution for the respective problems are equivalent. This can be noticed by viewing weak solutions

for both formalisms as a sequence of tasks that can be executed for favourable nondeterministic effects. Given a sequence, a policy can be formed for either formalism. $\square$

We notice that strong and strong cyclic solutions collapse for primitive problems as the same task network cannot be reached more than once due to the absence of methods. Then we get our complexity for primitive problems from Prop. 1 above and Thm. 4.8 and 5.1 by Chen and Bercher (2021).

Corollary 1. *Let $\mathcal{P}$ be a partially (totally ordered) primitive $FOND^{MP}$ HTN problem. Deciding whether $\mathcal{P}$ has a strong or strong cyclic solution is PSPACE-complete (in P).*

Proposition 2. *The definitions for weak $FOND^{MP}$ HTN and weak $FOND^{FM}$ HTN planning are equivalent.*

Proof. This can be realised by noticing that we can choose the methods corresponding to a trace of a weak $FOND^{MP}$ HTN solution for a weak $FOND^{FM}$ HTN solution. Conversely, we can construct a weak $FOND^{MP}$ HTN solution by first expanding the methods for a weak $FOND^{FM}$ HTN solution and then creating a policy corresponding to a $FOND^{FM}$ HTN policy for the expanded primitive task network. $\square$

As a direct consequence of this, we have that the complexity for weak $FOND^{MP}$ HTN planning is equivalent to that of weak $FOND^{FM}$ HTN planning by using Thms. 4.1/4.2/4.4/4.5 by Chen and Bercher (2021).

For acyclic problems, we again exploit the fact that since we will never reach the same task network twice under progression due to the absence of recursion in compound task decomposition, strong and strong cyclic solutions collapse. Although the term acyclic was originally intended to describe acyclicity of compound task decomposition in the deterministic HTN setting, it also happens to be the case that $FOND^{MP}$ HTN solutions themselves are acyclic.

Theorem 1. *Let $\mathcal{P}$ be a totally ordered acyclic $FOND^{MP}$ HTN problem. Deciding whether $\mathcal{P}$ has a strong or strong cyclic solution is EXPTIME-complete.*

Proof. Membership: we show that our progression algorithm (Alg. 2) always terminates and requires only polynomial space. We exploit the fact that for acyclic problems we can never reach the same task network and state pair more than once during progression. This means that strong and strong cyclic solutions coincide and that eventually all search nodes will have an empty task network or a rejecting state. Furthermore, we do not need the variable V to store the progression history. This leaves us with variables $tn, s, \mathcal{M}$. Clearly, s and $\mathcal{M}$ are polynomially bounded. The size of tn under progression is bounded in the same way as progression in the deterministic setting given that decomposition of compound tasks are equivalent. Thus, we can use Lemma 3.6 by Alford, Bercher, and Aha (2015) for totally ordered acyclic problems to get that tn is bounded polynomially. Hence, we have that totally ordered acyclic strong/strong cyclic $FOND^{MP}$ HTN planning is in $APSPACE = EXPTIME$ (Chandra and Stockmeyer 1976).

Hardness: we outline how to give a polynomial reduction of deciding whether an arbitrary ATM accepts an input string

w_I in space k . This will give us $\text{APSPACE} = \text{EXPTIME}$ -hardness. The main idea of the reduction is that we exploit the fact that a polynomially space bounded ATM can be decided in an exponential number of steps given that there are at most exponentially many configurations $C = |Q| \cdot (|\Gamma| + 1)^k \cdot k$ using k space. We get $|Q|$ from the number of states, $(|\Gamma| + 1)^k$ from all possible length k strings that can be constructed with the alphabet Γ plus a blank symbol, and k for the number of locations the tape head can be. We will define primitive tasks to mimic ATM transitions and use a task hierarchy to define a task network that can be decomposed into exponentially many tasks.

To model ATM configurations and transitions, we first define facts that represent the tape contents and ATM states with the same state variables as those in the PSPACE-hardness proof of non-hierarchical planning (Bylander 1994). We also define similar actions with the modification where given a $\forall$ state and an ATM transition, we create a nondeterministic task with the same number of corresponding effects, whereas in an $\exists$ state, we create a deterministic task modelling each effect. This enforces that at a $\forall$ state, all the next configurations must be accepting whereas at an $\exists$ state, we only have to choose one good effect.

To model exponentially many tasks, we construct compound tasks and methods in the same way as in Section 4 by Alford, Bercher, and Aha (2015). For ease of notation, let n be the smallest number such that $2^n \geq C$, the number of configurations shown to be exponential above. The main idea of the construction is that we define compound tasks $2^k \cdot \text{sim}$ for $0 < k \leq n$, each with one method which decomposes it into a totally ordered task network with two tasks mapping to $2^{k-1} \cdot \text{sim}$. Next, we have $1 \cdot \text{sim}$ have one method for each primitive task n decomposing it to $\text{tn}(n)$, and one method decomposing it to $\text{tn}_\emptyset$. In this way, we can define an initial task network $\text{tn}(2^n \cdot \text{sim})$ which can decompose into up to any number of tasks bounded exponentially to simulate an accepting ATM computation as required. $\square$

Proving EXPSPACE-hardness using ATMs is not as straightforward anymore as our reduction now has to be logarithmic. Thus, we can no longer define a fact for each tape cell which would cause a polynomial reduction and instead we will extend the NEXPTIME-hardness proof for deterministic acyclic HTN planning (Alford, Bercher, and Aha 2015) from the reduction of a nondeterministic Turing machine (NTM) to a reduction of an ATM. The idea of the original proof is that we do not define explicit facts to represent an NTM configuration but instead we only have one state and tape cell fact true at any time. Totally ordered primitive tasks are used to represent a witness and use synchronisation techniques to model and verify NTM transitions. This is because we can compactly represent k tasks in a task network with only a logarithmic number of defined tasks.

Theorem 2. *Let $\mathcal{P}$ be an acyclic FOND^{MP} HTN problem. Deciding whether $\mathcal{P}$ has a strong or strong cyclic solution is EXPSPACE-complete.*

Proof. Membership: we again use our alternating progression algorithm (Alg. 2) and the fact that strong and strong

cyclic solutions collapse for acyclic problems. Also similarly to the proof of membership of Theorem 1, we do not require the variable V although this is not necessary now as we will provide a time bound. We notice that the initial task network can be decomposed into a primitive task network with bounded size m^{k+1} where k is the maximum stratification of the compound tasks, and m is the size of the largest task network in the problem as shown in Corollary 3.2. by Alford, Bercher, and Aha (2015). Thus, the progression algorithm always terminates and determines if a solution exists within an exponential number of steps as the number of methods which can be applied is bounded exponentially and similarly with the execution of primitive tasks. So the problem is in $\text{AEXPTIME} = \text{EXPSPACE}$.

Hardness: we outline how to give a logarithmic reduction of deciding whether an arbitrary ATM A accepts a string w_I in time k . We extend the proof of NEXPTIME-hardness for deterministic acyclic problems in Thm. 6.1 by Alford, Bercher, and Aha (2015) which involved a reduction from an NTM. The idea of the original proof is to represent a witness for an NTM or a sequence of strings $w_0, \dots, w_k$ with exponentially many totally ordered tasks as described in the proof of Thm. 1 above. Each task asserts a fact representing a tape symbol at a tape cell, of which only one can be true at a given time. A second sequence of totally ordered tasks are synchronised with these tasks to check whether the i th character of strings w_j and w_{j+1} are equivalent. A third totally ordered sequence tracks the tape head and determine transitions. Thus, the transformed problem is able to generate any witness for the NTM with input w_I and yields a solution only if the witness is a proof for the original problem.

The modification we describe is by introducing additional nondeterministic tasks to model ATM transitions. In the original proof, there exist deterministic $\text{step}_\exists$ tasks for each nondeterministic effect of each nondeterministic transition in the first totally ordered sequence of tasks. We can introduce additional nondeterministic $\text{step}_\forall$ tasks which model universal ATM transitions. To make this work, we have the additional ATM assumption that at a given universal state, all transitions step in the same direction in order for the synchronisation process to still work. This can be compiled away by introducing additional states and deterministic transitions which take an ATM state back to its intended position after every universal transition. The correspondence of solutions still holds as a strong solution holds iff a computation tree exists for A determining that the initial configuration is accepting. This comes from being able to dynamically choose the correct decompositions in the second and third task sequences for verifying a computation tree induced by the first sequence of tasks. Thus, the problem is $\text{AEXPTIME} = \text{EXPSPACE}$ -hard and complete. $\square$

For regular problems, we exploit the fact that FOND^{MP} HTN planning is able to model non-hierarchical nondeterministic planning whose complexity we know. The idea of the reduction is that we can define a compound task which can decompose into arbitrarily many primitive tasks corresponding to actions for a non-hierarchical planning problem.

Theorem 3. Let $\mathcal{P}$ be a regular totally or partially ordered FOND^{MP} HTN problem. Deciding whether $\mathcal{P}$ has a strong or strong cyclic solution is EXPTIME-complete.

Proof. Membership: we use the bounded graph search algorithm described in the Search Algorithms section and recall that the set of reachable task networks for regular problems is bounded exponentially. This is also true for the number of states such that the size of S and hence the size of the problem to solve is exponential. Since the subroutine to solve strong or strong cyclic plan existence is polynomial with respect to the size of the graph, the problem is in EXPTIME.

Hardness: we model non-hierarchical nondeterministic planning problems with regular FOND^{MP} HTN problems in the same way described for the deterministic case (Erol, Hendler, and Nau 1996). The main idea is that we create a compound task *repeat* which has a method for every action in the original problem decomposing into a totally ordered task network with a task corresponding to such action followed by *repeat*. We also add a task *done* with precondition the goal condition. The only extension from the original proof is that we are able to model nondeterministic actions using nondeterministic tasks. The reduction mimics the mechanics of the original problem so strong and strong cyclic solutions correspond. It was shown by Rintanen (2004) that plan existence for both strong and strong cyclic planning is EXPTIME-complete by reduction from ATMs. Hence, the problem in question is EXPTIME-hard and complete. $\square$

Theorem 4. Let $\mathcal{P}$ be a totally ordered tail-recursive FOND^{MP} HTN problem. Deciding whether $\mathcal{P}$ has a strong or strong cyclic solution is EXPTIME-complete.

Proof. Membership: we will again use the bounded graph search algorithm provided in the Search Algorithms section and show that it runs in exponential time. To do this, we show that the number of reachable task networks under progression is only exponential. First, we have from Lem. 3.6 by Alford, Bercher, and Aha (2015) that under progression of a totally ordered tail-recursive HTN problem $\mathcal{P}$, a task network is bounded polynomially by $m = k + r \cdot h$ for k initial tasks, r the largest number of tasks in any method for $\mathcal{P}$ and h the height of the stratification on compound tasks. Note that although we are in the nondeterministic setting now, the bounds calculated for deterministic HTN problem carry over as the decomposition mechanics are the same.

Thus, letting $n = |\mathcal{N}_P \cup \mathcal{N}_C|$ be the number of task names in $\mathcal{P}$, the number of reachable task networks is bounded exponentially by $\sum_{i=0}^m i^n \leq (m+1)m^n$. The sum arises from counting the number of task networks of size i for $0 \leq i \leq n$ and the i^n from choosing any of n task names for each task in a totally ordered task network. Hence, the graph we build in the search algorithm has at most $n^m \cdot 2^{|\mathcal{F}|}$ nodes. Building and searching the graph takes exponential time, and thus the algorithm runtime.

Hardness: Given that regular problems are a special case of tail-recursive problems and that totally ordered regular strong and strong cyclic HTN planning is EXPTIME-complete, we have EXPTIME-hardness for totally ordered tail-recursive strong and strong cyclic HTN planning. $\square$

Theorem 5. Let $\mathcal{P}$ be a tail-recursive FOND^{MP} HTN problem. Deciding whether $\mathcal{P}$ has a strong or strong cyclic solution is in 2-EXPTIME. Determining existence of a strong cyclic solution for $\mathcal{P}$ is EXPSPACE-hard and a strong solution is 2-EXPTIME-hard and hence complete.

Proof. Membership: Again we use the bounded graph search algorithm but now the upper bound for reachable task networks is higher given that there is no longer any total order assumption. From Lemma 3.4 by Alford, Bercher, and Aha (2015), now the size of a task network under progression is bounded exponentially by $m = k \cdot r^h$ with variables the same as described in Thm. 4. Thus, letting n be the number of task names, the number of reachable task networks is upper bounded by $|TN_R| \leq \sum_{i=0}^m i^n \cdot f(i)$, where $f(i)$ counts the number of directed acyclic graphs for i labelled vertices. Again, i^n gives a loose upper bound for calculating the number of reachable non ordered task networks of size i , and the function f then gives us the number of possible partial orderings for size i task networks with names attached given that partial orderings are synonymous with directed acyclic graphs. We can provide a loose upper bound for the number of DAGs by counting the number of directed graphs: $f(n) \leq \sum_{i=0}^{n^2} \binom{n^2}{i} = 2^{n^2}$. This follows by noticing that there are at most n^2 directed edges for n vertices and that there are $\binom{n^2}{i}$ ways of choosing i edges for building a directed graph with i edges. Thus the number of reachable task networks is bounded by $|TN_R| \leq \sum_{i=0}^m i^n \cdot 2^{i^2} \leq (m+1) \cdot m^n \cdot 2^{m^2}$. Since m is exponential, the size of the graph is bounded double exponentially. Thus, the algorithm takes double exponential time to run.

Hardness: For strong cyclic problems, this follows from the fact that the deterministic version of the problem is EXPSPACE-complete (Alford, Bercher, and Aha 2015) and is a special case of nondeterminism. For strong problems, we extend the EXPSPACE-hardness proof for deterministic tail-recursive problems in the same fashion as described in Thm. 2 to get $\text{AEXPSPACE} = 2\text{-EXPTIME-hardness}$. $\square$

Conclusion

We revealed limitations of a previously introduced formalisation for HTN planning for fully observable environments and actions with nondeterministic effects. There, one has to find a primitive refinement first and then deal with action outcome uncertainty later on (FOND^{FM} HTN planning). We showed that there do exist solutions to such problems that can only be found when the choice of decomposition methods is delayed after the execution of nondeterministic actions (FOND^{MP} HTN planning).

For this new formalism we provided a complexity investigation that shows that FOND^{MP} HTN is almost always one class harder than the most expressive variant of FOND^{FM} HTN planning for totally ordered problems. For the partially ordered variants we cannot draw a clear conclusion since many results for FOND^{FM} HTN planning are not yet tight. We have also provided two search algorithms to aid with membership proofs but also to serve as baseline methods for future implementations.

References

- Alford, R.; Bercher, P.; and Aha, D. 2015. Tight Bounds for HTN Planning. In *ICAPS 2015*, 7–15. AAAI Press.
- Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2012. HTN Problem Spaces: Structure, Algorithms, Termination. In *SOCS 2012*. AAAI Press.
- Babai, L.; Kantor, W. M.; and Luks, E. M. 1983. Computational complexity and the classification of finite simple groups. In *24th Annual Symposium on Foundations of Computer Science (SFCS 1983)*, 162–171. IEEE.
- Babai, L.; and Luks, E. M. 1983. Canonical labeling of graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, 171–183.
- Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *IJCAI 2019*, 6267–6275. IJCAI.
- Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1-2): 165–204.
- Chandra, A. K.; and Stockmeyer, L. J. 1976. Alternation. In *17th Annual Symposium on Foundations of Computer Science (FOCS 1976)*, 98–108. IEEE.
- Chen, D.; and Bercher, P. 2021. Fully Observable Nondeterministic HTN Planning – Formalisation and Complexity Results. In *ICAPS 2021*, 74–84. AAAI Press.
- Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2): 35–84.
- Erol, K.; Hendler, J.; and Nau, D. S. 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 *IJCAI 2011*, 1955–1961. AAAI Press.
- Goldman, R. P.; and Kuter, U. 2019. Hierarchical Task Network Planning in Common Lisp: the case of SHOP3. In *Proceedings of the 12th European Lisp Symposium (ELS 2019)*, 73–80. ELSAA.
- Höller, D.; and Behnke, G. 2021. Loop Detection in the PANDA Planning System. In *ICAPS 2021*, 168–173. AAAI Press.
- Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2018. A Generic Method to Guide HTN Progression Search with Classical Heuristics. In *ICAPS 2018*, 114–122. AAAI Press.
- Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020. HTN Planning as Heuristic Progression Search. *JAIR*, 67: 835–880.
- Kuter, U.; and Nau, D. S. 2004. Forward-Chaining Planning in Nondeterministic Domains. In *AAAI 2014*, 513–518. AAAI Press / The MIT Press.
- Kuter, U.; Nau, D. S.; Pistore, M.; and Traverso, P. 2005. A Hierarchical Task-Network Planner based on Symbolic Model Checking. In *ICAPS 2015*, 300–309. AAAI.
- Kuter, U.; Nau, D. S.; Pistore, M.; and Traverso, P. 2009. Task decomposition on abstract states, for planning under nondeterminism. *Artificial Intelligence*, 173(5-6): 669–695.
- Li, R.; Patra, S.; and Nau, D. 2021. Decentralized Refinement Planning and Acting. In *Proceedings of the 4th ICAPS Workshop on Hierarchical Planning (HPlan 2021)*, 225–233. AAAI Press.
- McKay, B. D.; and Piperno, A. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60: 94–112.
- Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Muñoz-Avila, H.; Murdock, J. W.; Wu, D.; and Yaman, F. 2005. Applications of SHOP and SHOP2. *IEEE Intell. Syst.*, 20(2): 34–41.
- Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *JAIR*, 20: 379–404.
- Nau, D. S.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple Hierarchical Ordered Planner. In *IJCAI 99*, 968–975. Morgan Kaufmann.
- Patra, S.; Mason, J.; Ghallab, M.; Nau, D.; and Traverso, P. 2021. Deliberative acting, planning and learning with hierarchical operational models. *Artificial Intelligence*, 299: 103523.
- Patra, S.; Mason, J.; Kumar, A.; Ghallab, M.; Traverso, P.; and Nau, D. S. 2020. Integrating Acting, Planning, and Learning in Hierarchical Operational Models. In *ICAPS 2020*, 478–487. AAAI Press.
- Richter, F.; and Biundo, S. 2017. Addressing Uncertainty in Hierarchical User-Centered Planning. In *Companion Technology, Cognitive Technologies*, 101–121. Springer.
- Rintanen, J. 2004. Complexity of Planning with Partial Observability. In *ICAPS 2004*, 345–354. AAAI.