Hierarchical Task Network Planning with Task Insertion and State Constraints
Zhanhao Xiao, Andreas Herzig, Laurent Perrussel, Hai Wan, Xiaoheng Su

To cite this version:
Zhanhao Xiao, Andreas Herzig, Laurent Perrussel, Hai Wan, Xiaoheng Su. Hierarchical Task Network Planning with Task Insertion and State Constraints. 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), Aug 2017, Melbourne, Australia. pp.4463-4469. hal-03658077

HAL Id: hal-03658077
https://hal.science/hal-03658077
Submitted on 3 May 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author's version published in: 
http://oatao.univ-toulouse.fr/19161

Official URL


Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr
Hierarchical Task Network Planning with Task Insertion and State Constraints

Zhanhao Xiao1,2, Andreas Herzig1,3, Laurent Perrussel1, Hai Wan4,5, and Xiaoheng Su1

1IRIT, University of Toulouse, Toulouse, France
2AIRG, Western Sydney University, Penrith, Australia
3IRIT, CNRS, Toulouse, France
4School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
5Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou, China

zhanhao.xiao@ut-capitole.fr

Abstract

We extend hierarchical task network planning with task insertion (TIHTN) by introducing state constraints, called TIHTNS. We show that just as for TIHTN planning, all solutions of the TIHTNS planning problem can be obtained by acyclic decomposition and task insertion, entailing that its plan-existence problem is decidable without any restriction on decomposition methods. We also prove that the extension by state constraints does not increase the complexity of the plan-existence problem, which stays \(2\text{-NEXPTIME}\)-complete, based on an acyclic progression operator. In addition, we show that TIHTNS planning covers not only the original TIHTN planning but also hierarchy-relaxed hierarchical goal network planning.

1 Introduction

Hierarchical task network (HTN) planning [Erol et al., 1994] is an approach for building plans via step-wise refinement of high-level tasks into lower-level tasks in a top-down manner. A task network may contain both compound (high-level) and primitive tasks. Primitive tasks correspond to STRIPS-like actions that can be applied in states where their preconditions are met, while compound tasks are abstractions: for every compound task, the domain features a set of decomposition methods, each mapping the task to a task network. The complexity of the plan-existence problem for HTN planning ranges up to undecidability even for propositional HTN planning [Erol et al., 1994]. Even so, hierarchical planning approaches are often chosen for real world application scenarios [Lin et al., 2008; Biundo et al., 2011]. On the other hand, it is usually a challenge to provide a complete domain which includes all possible methods for all compound tasks, while defining only a partially hierarchical domain is not sufficient to produce all desired solutions. Several HTN researchers have investigated how partially hierarchical domain knowledge can be exploited during planning without relying on the standard HTN formalism [Kambhampati et al., 1998; Biundo and Schattenberg, 2001; Geier and Bercher, 2011; Shivashankar et al., 2013]. Among them, hierarchical task network with task insertion (TIHTN) planning [Geier and Bercher, 2011] relaxes the restriction on solutions and allows solutions generated not only by the decomposition of compound tasks, but also by the insertion of primitive tasks from outside the decomposition hierarchy.

Unfortunately, only ordering constraints are considered in propositional TIHTN planning [Geier and Bercher, 2011] and lifted TIHTN planning [Alford et al., 2015b]. In contrast, state constraints are taken into account in the conventional HTN planning [Erol et al., 1994], but there is few work on considering them in TIHTN planning.

In this paper, we investigate the extension of TIHTN with state constraints, noted TIHTNS for short. We first prove that, just as for the TIHTN planning problem, all solutions of the TIHTNS problems can be obtained by acyclic decomposition and task insertion, entailing that it is decidable without any restrictions on decomposition methods. We then show that the extension by state constraints does not cause an increase in complexity of the plan-existence problem, which stays \(2\text{-NEXPTIME}\)-complete, based on an acyclic progression operator. We also show that TIHTNS planning includes lifted TIHTN planning. As under task insertion semantics, hierarchical goal network (HGN) planning [Shivashankar et al., 2013] and goal-task network (GTN) planning [Alford et al., 2016] can be translated to lifted TIHTN, our framework, TIHTNS, actually covers the two kinds of planning approaches. In addition, we give an alternative embedding of hierarchy-relaxed HGN (HR-HGN) [Shivashankar et al., 2017] in TIHTNS without introducing fresh operators.

State constraints State constraints can capture the pre- and postcondition of compound tasks, though in the standard HTN planning there is no notion of pre- and postcondition of compound tasks. A compound task is considered as accomplished if its subtasks are accomplished. With a state constraint, a formula, as a postcondition, can be required to hold after accomplishing a compound task.

Ordering constraints cannot fully represent state constraints. The ‘immediate’ state constraint, which requires a formula holds immediately before or after a compound task, can be simulated via introducing a virtual subtask to check whether the formula holds. However, the ‘maintenance’ state constraints (also called trajectory constraints in the Planning Domain Definition Language 3 (PDDL3)) [Gerevini and Long, 2005] cannot be represented easily. For instance, suppose a robot is required to always keep 10% battery for emergency. Its initial compound task is to “clean a room”, de-
First, we define a function-free first order language i.e.,
add "but not" 
Opened(a) 
Opened(b) 
positive effect "being 
", after grounding 
openDoor(a) 
neously . For example, the positive effect of " 
variables in the operator so that they are instantiated simulta-
its domain is 
where

Furthermore, state constraints are introduced into PDDL3 [Gerevini and Long, 2005] to support hard constraints over state properties of a trajectory and the specification of preferences. Later [Sohrabi et al., 2009] extends PDDL3 into HTN planning where state constraints are used to capture user preferences. State constraints are also necessary in real-world applications, such as in web service composition where state constraints are used to describe user preferences [Lin et al., 2008] and to additionally capture the enforcement of regulations [Sohrabi and McIlraith, 2009].

The rest of the paper is structured as follows. Section 2 presents the definition of TIHTNS. Section 3 presents how to embed lifted TIHTN and HR-HGN into TIHTNS. Section 4 presents the complexity results of TIHTNS. Section 5 concludes and discusses the future work.

## 2 Extended TIHTN Planning

In this section we adapt the original TIHTN planning formalism of [Geier and Bercher, 2011; Alford et al., 2015b]. First, we define a function-free first order language $\mathcal{L}$ from a set of variables and a finite set $\mathcal{L}_0$ of predicates and constants. Next we take parts of variables in $\mathcal{L}$ as task symbols to identify tasks. Every task is associated with an action (task name), which is syntactically a first-order atom in $\mathcal{L}$. That is, every action typically is associated with an arity and contains variables that can be eliminated via grounding. For example, by grounding the action "openDoor(X)" where $X$ is quantified as a door, we can obtain a set of ground actions "openDoor(a)", "openDoor(b)", etc. Those actions which can be executed directly are called operators, noted $\mathcal{O}$, while others are called compound actions,

An HTN problem is a tuple $(T, \Delta, \alpha)$, where $T$ is a finite and non-empty set of task symbols; $\Delta \subseteq (T \cup \{\text{nil}\}) \times \mathcal{L} \times (T \cup \{\text{nil}\})$ is a set of constraints over $T$; $\alpha : T \rightarrow \mathcal{C} \cup \mathcal{O}$ labels every task with an action.

### 2.1 HTN Problems

**Task networks** A task network $tn = (T, \Delta, \alpha)$ is a tuple, where
- $T$ is a finite and non-empty set of task symbols;
- $\Delta \subseteq (T \cup \{\text{nil}\}) \times \mathcal{L} \times (T \cup \{\text{nil}\})$ is a set of constraints over $T$;
- $\alpha : T \rightarrow \mathcal{C} \cup \mathcal{O}$ labels every task with an action.

With Function $\alpha$, we allow multiple instances of an action in a task network. Compared to the ordering constraints in [Geier and Bercher, 2011] which are in form of task-task pairs, we use a triple $(t_i, \varphi, t_j)$ to denote a state constraint which intuitively means that formula $\varphi$ must be true in all states between $t_i$ and $t_j$. Specially, we introduce an idle task symbol $\text{nil}$ which designates a task that is accomplished immediately: $(\text{nil}, \varphi, t_j)$ and $(t_i, \varphi, \text{nil})$ mean formula $\varphi$ holds immediately before $t_i$ and after $t_j$, respectively. We suppose $\text{nil}$ only occurs in constraints. When $\varphi$ is the truth constant $\top$ then the state constraint $(t_i, \varphi, t_j)$ becomes an ordering constraint that just requires that $t_i$ is before $t_j$. A task is primitive if it is associated to an operator, otherwise compound. A task network is primitive if it only contains primitive tasks.

We say that two task networks $tn = (T, \Delta, \alpha)$ and $tn' = (T', \Delta', \alpha')$ are isomorphic, noted $tn \cong tn'$, if there exists a bijection $\delta : T \rightarrow T'$ where for all $t, t' \in T$ it holds that $\alpha(t) = \alpha'(\delta(t))$ and $(t, \varphi, t') \in \Delta$ iff $(\delta(t), \varphi, \delta(t')) \in \Delta'$.

**Methods** Non-primitive task networks contain compound tasks which cannot be executed directly by the agent, and decomposition methods tell us how to decompose these hierarchically. Each decomposition method $m$ is a tuple $(c, tn_m)$, where $c$ is a compound action, called the method’s head, and $tn_m$ is a task network, whose inner tasks are called the method’s subtasks. The intuition is that compound action $c$ can be reduced by the subtask network $tn_m$.

**HTN problems** TIHTNS problems only differ HTN problems in the solution criterion and share the syntactical problem description. An HTN domain is a tuple $D = (\mathcal{L}, \mathcal{C}, \mathcal{O}, M)$ where $M$ is a set of methods and $\mathcal{C} \cap \mathcal{O} = \emptyset$. An HTN problem is a tuple $P = (D, s_I, tn_I)$ where $s_I$ is the ground initial state and $tn_I$ is the initial task network.

The semantics of HTN planning is given through grounding. According to [Alford et al., 2015a], it is easy to translate a lifted HTN problem into a ground (or propositional) HTN problem as the set of relations and constants $\mathcal{L}_0$ is finite. For an HTN problem $(\mathcal{L}, \mathcal{C}, \mathcal{O}, M, s_I, tn_I)$, we use $P = (\mathcal{L}, \mathcal{C}, \mathcal{O}, M, s_J, tn'_I)$ to denote the ground (or propositional) problem obtain from it.

A ground state is a subset of the ground atoms in $\mathcal{L}_0$. A set of ground operators $\mathcal{O}$ determines a state-transition function $\gamma : 2^{\mathcal{L}_0} \times \mathcal{O} \rightarrow 2^{\mathcal{L}_0}$, where:

- $\gamma(s, o)$ is defined iff $s \models \text{pre}(o)$;
- $\gamma(s, o) = (s \setminus \text{del}(o)) \cup \text{add}(o)$ if $\gamma(s, o)$ is defined.
A sequence of operators \( \langle o_1, \ldots, o_n \rangle \) is executable in a state \( s_0 \) iff there exists a sequence of states \( s_1, \ldots, s_n \) such that \( \forall 1 \leq i \leq n \), \( (s_{i-1}, o_i) = s_i \).

**Example 1.** Suppose we have an initial compound action \( \text{goMC} \) for “go to Melbourne center”, operator \( o_1 \) for “fly to Melbourne”, and operator \( o_2 \) for “take a taxi to the center”. We use \( \text{At(Mc)} \) and \( \text{At(Ma)} \) to denote “being at Melbourne center” and “being at Melbourne airport”. The primitive operators are:

- \( o_1 = (T, \{\text{At(Ma)}\}, \emptyset) \)
- \( o_2 = (\text{At(Ma)}, \{\text{At(Mc)}\}, \{\text{At(Ma)}\}) \)

and the method is:

\[ m = (\text{goMC}, \text{tn}_{\text{m}}), \text{where } \text{tn}_{\text{m}} = (t_2, \emptyset, (t_2, o_1)) \]

Then the HTN problem \( P \) is \( (L, \text{goMC}, \{o_1, o_2\}, m, s_1, \text{tn}_j) \) with \( s_1 = \emptyset \) and \( \text{tn}_j = (t_1, (t_1, \text{At(Mc)}, \text{nil}), (t_1, \text{goMC})) \).

### 2.2 Task Decomposition

Next we borrow the notion of decomposition in [Geier and Bercher, 2011] and define how to decompose a compound task into a task network. In order to indicate the starting point and the end point of a compound task \( t \), we introduce a pair of ‘dummy’ primitive tasks, noted \( st \) and \( ts \). As nil only occurs in constraints, we suppose the restriction of constraint set \( \Delta \) to a set of tasks \( T \) is \( \Delta|_T = \Delta(n) \cap L \subset X(T \cup \{\text{nil}\}) \).

**Definition 1 (Decomposition).** Given an HTN domain \( D = (L, C, O, M) \), let \( t_n = (T, \Delta, \alpha) \) be a task network and \( t \in T \) be a compound task. Let \( m = (\alpha(t), (T_m, \Delta_m, o_m)) \) and \( T_m \cap T = \emptyset \). The decomposition of task \( t \) by method \( m \) is \( \text{tn} = (T', \Delta', \alpha') \) where

\[ T' = (T \setminus \{t\}) \cup T_m \cup \{st, ts\} \]

\[ \Delta' = \Delta|_{T'} \cup \Delta_m \cup \{(st, t, j), (t, j, ts) \mid t_j \in T_m\} \]

\[ \cup \{(t* \varphi, ts) \mid (t, \varphi, t) \in \Delta\} \]

\[ \cup \{(t*, \varphi, t_2) \mid (t, \varphi, t_2) \in \Delta\} \]

\[ \alpha' = \alpha|_{T'} \cup o_m \cup \{(st, \text{skip}), (ts, \text{skip})\} \]

We write \( \text{tn} \rightarrow_{t,m} \text{tn}' \) when \( \text{tn}' \) is the decomposition of \( t \) by \( m \).

In the resulting task network, the decomposed compound task is replaced with subtasks defined by the method applied and its corresponding starting and terminating tasks. The latter two are dummy tasks and are mapped to the action skip. All state constraints about the decomposed task \( t \) are propagated by \( st \) and \( ts \) in \( \Delta' \). More precisely, if \( \varphi \) holds before \( t \) then it also holds before \( st \) and if \( \varphi' \) holds after \( t \) then it also holds after \( ts \). Subtasks should satisfy the inner constraints introduced by the decomposition method and should be performed between \( st \) and \( ts \).

**Example 2 (Example 1 continued).** We apply the method \( m \) in \( \text{tn}_j \) to decompose \( t_1 \), i.e., \( \text{tn}_j \rightarrow_{t_1,m} \text{tn}' \), where \( \text{tn}' \) is:

\[ T' = \{t_2, st_1, t_1\} \]

\[ \Delta' = \{(t_2, T, t_1), (st_1, T, t_2), (t_1*, \text{At(Mc)}, \text{nil})\} \]

\[ \alpha' = \{(t_2, o_1), (st_1, \text{skip}), (t_1*, \text{skip})\} \]

### 2.3 Solutions

A solution of an HTN problem is a sequence of primitive tasks which is also called a plan of the problem.

**Consistency with constraints**

Given a primitive task network \( \text{tn} = (T, \Delta, \sigma) \) where \( |T| = n \), let \( \sigma : T \rightarrow \{1, \ldots, n\} \) be a bijection. We use \( \sigma \) to form a total ordering, noted \( \sigma(\text{tn}) \), of tasks in \( T \) as: \( \sigma^-(1), \ldots, \sigma^-(n) \) where \( \sigma^- \) is the inverse function of \( \sigma \), i.e., \( \sigma^-\sigma(-1) = t \). Suppose \( \alpha(\sigma(\text{tn})) \) is executable in \( s_0 \), i.e., there exists a sequence \( s_1, \ldots, s_n \) such that \( \gamma(s_{i-1}, \alpha(t_i)) = s_i \) for every \( i \) such that \( 1 \leq i \leq n \). We say that \( \sigma(\text{tn}) \) is consistent with \( \Delta \) in \( s_0 \) if for every \( \sigma^-i, \sigma^-j \in T \) the following hold:

- for every \( (\text{nil}, \varphi, \sigma^-i) \in \Delta \), \( s_{i-1} = \varphi \);
• for every \((σ^- (i), φ, nil) ∈ Δ, s_j → φ;\)
• for every \((σ^- (i), φ, σ^-(j)) ∈ Δ, i < j \) and \(s_j → φ;\)
for every \(i ≤ k < j\).

Intuitively, a ‘maintenance’ state constraint \((t, φ, t')\) is satisfied if all states between \(t\) and \(t'\) satisfy \(φ\). An ‘immediate’ state constraint \((nil, φ, t)\) is satisfied if \(φ\) holds in the state right before \(t\) occurs (or \(t\) is a primitive task), in other words right before all subtasks of \(t\). Finally, an ‘immediate’ state constraint \((t, φ, nil)\) is satisfied if \(φ\) holds right after the state where \(t\) (or \(t\) primitive task) occurs, in other words right after all subtasks of \(t\). Note that it is impossible to satisfy \((t, ⊥, t')\) because there is no state \(s\) such that \(s \models ⊥\).

**Executability** A task network \(tn\) is primitive iff it contains only primitive tasks. A primitive task network \(tn\) is executable in a state \(s\) if there exists a total ordering \(σ(tn)\) of the tasks in \(tn\) that is consistent with \(Δ\) in \(s\). We called such a \(σ(tn)\) a plan of \(tn\) in \(s\), noted \(σ_{tn,s}\).

It is possible that the task network is not executable in a state. For instance, the primitive task network \(tn\) in Example 2 is not executable in \(s_j\) because apart from skip it only involves the operator \(o_1\) and it is impossible to satisfy \(At(Mc)\). However, if we extend the task network by inserting some tasks, then we can make it executable.

**Insertion** Let \(tn = (T, Δ, α)\) and \(tn' = (T', Δ', α')\) be two task networks where \(tn'\) is primitive. Inserting \(tn'\) into \(tn\) results in the task network \(tn_1 = tn \cup tn'\).

Note that it is not required that \(T' \cap T = \emptyset\) because the insertion can involve some constraints about tasks in \(tn\).

**Solutions** With respect to some initial \(tn\), if \(tn'\) is reachable by a finite sequence of decompositions and an insertion we write \(tn →^*_D tn'\). The plan obtained only by decompositions is called an HTN solution while the plan obtained additionally by insertion is called a TIHTNS solution.

Let \(tn\) be a primitive task network such that there exists a plan of \(tn\) in \(s_j\). Given an HTN problem \(P\), we call \(σ_{tn,s_j}\) an HTN solution of \(P\) if \(tn_1 →^*_D tn\); we call \(σ_{tn,s_j}\) a TIHTNS solution of \(P\) if \(tn →^*_D tn\).

**Example 3** (Example 2 continued). The plan \(*t_1, t_2, t_3, t_1*\) where \(α(t_3) = o_2\) is a TIHTNS solution of \(P\).

When considering whether an HTN problem \(P\) has a TIHTNS solution, we call \(P\) a TIHTNS problem.

The next proposition states that the state constraints about compound tasks are satisfied in the solutions of the problem.

**Proposition 2.** Given a TIHTNS problem \(P\), suppose \(Tr\) is a valid decomposition tree with respect to \(P\) and \(tn'\) is the final primitive task network obtained from \(ψ(Tr)\) by insertion and \(σ_{tn,s_j}\) is a solution of \(P\). Then all constraints in \(Tr\) are satisfied by \(σ_{tn',s_j}\).

**Sketch of proof.** Suppose there are two compound tasks \(t_i\) and \(t_j\) in \(Tr\) and \(tn' = (T', Δ', α')\). For every \((t_i, φ, t_j) \in Δ\), we have \((t_i, φ, t_j) \in Δ'\) and then all states \(s_k\) such that \(σ(t_i) ≤ k < σ(t_j), s_k \models φ\). For every \((t_i, φ, nil) \in Δ\), we have \((t_i, φ, nil) \in Δ'\) and then \(s(σ(t_i)) \models φ\). For all states \(σ(st) < σ(t_i)\) for all \(st \in sub(t_i)\). The case of \((nil, φ, t_j)\) is similar. If one or both of \(t_i, t_j\) are primitive then just consider \(*t_j, t_i*\) as \(t_i\) and \(t_j\), respectively. Then all constraints in \(Tr\) are satisfied.

**Remark 1.** The state constraint here we define is weaker than that in the original HTN planning [Erol et al., 1995]. Their semantics considers every compound task starts by its first ‘real’ subtask and terminates by its last ‘real’ subtask instead of the virtual starting and terminating tasks. The distinction between two semantics stands on whether it is allowed to insert tasks between \(t\) and the first real subtask of \(t\) (between the last real subtask and \(t\)). Our weaker semantics can capture the pre- and postcondition of compound actions better. In Example 1, after the subtask \(t_3\) of goMC is accomplished, the agent’s desirable goal “being in the center” does not hold while by allowing the insertion of \(t_3\), the goal is achieved.

### 3 Embedding TIHTN and HGN

Recently HTN researchers work on enhancing the semantics of HTN planning and have proposed variants of HTN planning. Propositional TIHTN was first proposed in [Geier and Bercher, 2011] and later was extended into lifted TIHTN in [Alford et al., 2015b]. Hierarchy goal network (HGN) planning [Shivashankar et al., 2012] operates over a hierarchy of goals with methods that decompose goals with further subgoals. [Shivashankar et al., 2017] relaxes the hierarchy of HGN planning and translates the variant which is called HR-HGN planning into classical planning. Hereafter, we detail that how lifted TIHTN and HR-HGN planning can be easily encoded in our framework in polynomial time.

**Embedding TIHTN** Let us now translate lifted TIHTN into our extended TIHTNs. By replacing all occurrences of \(t_i < t_j\) in lifted TIHTN problem \(P\) with \((t_i, T, t_j)\), we obtain a TIHTNS problem, noted \(Γ_T(P)\). The following proposition shows that the two problems are equivalent.

**Proposition 3.** For a lifted TIHTN problem \(P, P\) has a solution iff \(Γ_T(P)\) has a solution.

**Sketch of proof.** Under the lifted TIHTN semantics, consider the decomposition of \(t\) by \(m\), the new ordering constraint set is obtained as: (1) keep the constraints not involving \(t\); (2) introduce the constraints about the subtasks; (3) if \(t\) is before \(t'\) then all subtasks are before \(t'\); (4) if \(t\) is after \(t'\) then all subtasks are after \(t'\). Under the extended semantics, according to Definition 1, the change of state constraints includes (1) and (2) and is analogous to (3) and (4). If \(t\) is before \(t'\), i.e., \((t, T, t')\), there will be \((t*, T, t')\) introduced. As \((t_m, T, t_n)\) for all subtasks \(t_m\) are introduced, it entails that \((t_m, T, t')\) which means \(t_m\) is before \(t'\). Then (3) is simulated and the case of (4) is similar. The proposition follows. 

Alford et al. [2016] combine HTN and HGN planning into goal-task network (GTN) planning where an element of the network consists of a goal and a task. They also show that allowing task insertion, HGN and GTN planning problems can be translated polynomially into lifted TIHTN problems. Therefore, our extension TIHTNS actually also covers them.

**Embedding HR-HGN** Hierarchical goal network planning is a formalism which extends classical planning to include hierarchical decomposition using methods.

HGN planning talks about goal network \(gn = (G, ψ)\) where \(G\) is a set of formulas in disjunctive normal form over ground literals, called goal formulas, and \(ψ\) is a
strict partial order on $G$. Similar with HTN problems, an HGN problem is a tuple $P = (\mathcal{O}, M, s_I, gn_I)$ where $M$ is a set of HGN methods and $gn_I$ is an initial goal network. For an HR-HGN problem, $M$ is an empty set and omitted. The solutions of HR-HGN are defined as the set of all operators sequences that are executable in the initial state $s_I$ and that achieve all initial goals according to the order. In [Shivashankar et al., 2017], by introducing fresh operators with the number of $|gn_I|$, an H-R-HGN problem can be translated into a classical planning problem. Now, we translate HR-HGN planning into our framework without introducing any fresh operators. Allowing task insertion, state constraints are translated into a classical planning problem. Now, we translate HR-HGN planning into our framework without introducing any fresh operators. Allowing task insertion, state constraints are translated into a classical planning problem.

\[\text{Proposition 4. For an HGN problem } P, \ P \text{ has a solution iff } \Gamma_C(P) \text{ has a solution.}\]

\[\text{Sketch of proof. Suppose } \sigma_{w', s_I} \text{ is a solution of } \Gamma_C(P). \text{ Then } t' \text{ is obtained from } tn' \text{ by an insertion. It forms a sequence } s_0, \ldots, s_n \text{ where } s_0 = s_I. \text{ For every } g \in G_I, \text{ we have } \lambda_g \subseteq T' \text{ and (nil, } g, \lambda_g) \in \Delta' \text{ where } \sigma(\lambda_g) = t; \text{ for every } g_i < g_j, \text{ we have } (\lambda_{g_i}, T, \lambda_{g_j}) \Delta' \text{ then } i' < j' \text{ and } s_{i'} \models g, s_{j'} \models g, \text{ where } \sigma(\lambda_{g_i}) \text{ and } \sigma(\lambda_{g_j}).\]

4 Complexity

In this section, we show that the solutions of THHTNS can be obtained by acyclic decomposition and it does not increase the complexity, staying \textsc{2-Nexptime}-complete.

\[\text{Substitution} \]

Now we adapt the notion of subtree substitution initially proposed in [Geier and Bercher, 2011] which replaces a subtree with another subtree. Given a decomposition tree $(T, E, \Delta, \alpha, \beta)$ and a node $t \in T$, we define the subtree of Tr induced by $t$, as $\text{Tr}[t] = (T', E', \Delta', \alpha|_{T'}, \beta|_{T'})$ where $\Delta' = \Delta[T|E \setminus \{\{\text{nil, } \varphi_1, \varphi_2, \text{nil}\} \in \Delta\}$ and $\alpha|_{T'}$ and $\beta|_{T'}$ are defined as $\alpha|_{T'} = \alpha|_{T} \setminus \{\{\text{nil, } \varphi_1, \varphi_2, \text{nil}\} \in \Delta\}$ and $\beta|_{T'} = \beta|_{T} \setminus \{\{\text{nil, } \varphi_1, \varphi_2, \text{nil}\} \in \Delta\}$.

\[\text{Proposition 5. Let } Tr = (T, E, \Delta, \alpha, \beta) \text{ be a valid decomposition tree with respect to TIHTNS problem } P \text{ and two nodes } t_1, t_2 \in T, t_j \in \tau(\text{Tr}[t_i]) \text{ with } \alpha(t_i) = \alpha(t_j). \text{ Then Tr}[t_1 \leftarrow t_j] \text{ is also a valid decomposition tree w.r.t. } P.\]

\[\text{Acyclic decomposition} \]

The sequence of decompositions is acyclic if for every node $t$ in its corresponding tree $Tr$, the ancestors of $t$ have different actions.

The insertion of tasks allow us to break the loop of generating same compound tasks during the decomposition procedure and find a shortcut to generate the solution. The following theorem states that we only need to consider the acyclic sequences of decompositions to compute the solution.

\[\text{Theorem 1. A TIHTNS problem } P \text{ has a solution iff } P \text{ has a solution which is generated by an acyclic sequence of decompositions and an insertion.}\]

\[\text{Proof.} \quad \text{The right-to-left direction is straightforward. Now we prove the left-to-right direction.}\]

Suppose $\sigma_{tn, s_I}$ is a TIHTNS solution of $P$ where $tn \rightarrow_D t_{n+1}$ and $tn$ is obtained from $tn_1$ by insertion. Then $tn_1$ is a primitive task network. If the sequence of decompositions from $tn_1$ to $tn$ is acyclic, then it is proved. Now suppose the sequence is not acyclic. Then in its corresponding tree $Tr$, there exist two ancestors $t_i, t_j$ of some $t$ such that $\alpha(t_i) = \alpha(t_j)$. By Proposition 5, the tree $Tr[t_i \leftarrow t_j]$ is valid with respect to $P$. By Proposition 1, there exists a primitive task network $tn_2$ such that $tn_2 = \nu(Tr[t_i \leftarrow t_j])$ and $tn_1 \rightarrow_D t_{n+1}$. As the substitution does not introduce any new node, the leaf nodes of $Tr[t_i \leftarrow t_j]$ are a subset of the leaf nodes of $Tr$. The only state constraints about leaf nodes which are introduced by the substitution are $(t_{j*}, \varphi_1, t_{j*})$ and $(t_{j*}, \varphi_2, t_{j*})$. Now we replace all occurrences of $t_{j*}$ with $t_{j*}$ and $t_{j*}$ with $t_{j*}$ in $Tr[t_i \leftarrow t_j]$. Then except for those constraints between subtasks of $t_i$ and $t_j$, the case of $t_{j*}$ is handled in a similar way.

The next lemma allows us to only check whether there is a solution with an upper bound on length.

\[\text{Lemma 1. If TIHTNS problem } P = (\mathcal{L}, C, O, M, s_I, tn_1) \text{ has a solution then } P \text{ has a solution with a length of at most } |T|! \times (k+2)^{|C|} \times c^m, \text{ where } k \text{ is the maximal number of subtasks in one method, } c \text{ and } p \text{ are the number of constants and predicates respectively, } m \text{ and } n \text{ are the maximal arity of compound actions and predicates respectively.}\]

\[\text{Sketch of proof. A valid decomposition tree generated by acyclic decomposition has at most } |T|! \times (k+2)^{|C|} \text{ leaf nodes where } |C| \text{ is the number of ground compound actions that is } |C| \times c^m. \text{ As the number of states is bounded by } 2^{2^{|C|}}, \text{ where } |\mathcal{L}_0| \text{ is the number of ground atoms that is } p \times c^m, \text{ at most } 2^{2^{|C|}} \text{ tasks can be inserted between two neighbor tasks.}\]
As the state-transition function $\gamma$ satisfies the frame axiom, before performing an operator we can check whether a formula is true in the next state. Given a formula $\varphi$, we use $\varphi^o$ to denote the formula obtained from $\varphi$ by replacing $p$ with $\top$ and $q$ with $\bot$ where $p \in \text{add}(o)$ and $q \in \text{del}(o)$. As all variables outside the effect of operator $o$ keep their truth value, we can check whether a formula $\varphi$ holds in the next state before operator $o$ is performed, as the following lemma states:

**Lemma 2.** $s \models \varphi^o$ if $\gamma(s,o) \models \varphi$.

**Acyclic progression** We adapt the acyclic progression operator proposed in [Alford et al., 2015b] to TIHTNS. To capture the ‘maintenance’ state constraints we introduce a set $\Sigma$ of pairs $(\varphi,t)$ of formula and task which means $\varphi$ should be satisfied until performing task $t$. We use $\text{Fml}(\Sigma)$ to denote the conjunction of all formulas in $\Sigma$. To forbid the recursion of tasks, when decomposing a compound task $t$ by method $m$, its subtasks cannot contain $t$’s ancestors denoted by $h(t)$, where $h : T \rightarrow 2^T$. Formally, we use a tuple $(s, t, \Sigma, h)$ to represent that task network still needs to be accomplished at the current state $s$. Given a tuple $(s, t, \Sigma, h)$, we define its acyclic progression as $(s', t', \Sigma', h')$ with $t'n = (T', \Delta', \alpha')$, which is obtained from three possible options:

- **Task insertion:** if $s \models \text{pre(o)} \land \text{Fml}(\Sigma)^o$ then $s' = \gamma(s,o)$ and $t'n = t, \Sigma' = \Sigma, h' = h$.
  The task inserted should satisfy the state constraints.

- **Task performing:** for a primitive task $t \in T$, if $s \models \text{pre}(a(t)) \land \text{Fml}(\Sigma^{-}\alpha(t))$ where $\Sigma^{-}\alpha(t) = \Sigma \cap (\mathcal{L} \times (T \setminus \{t\}))$ and the followings hold:
  - there is no $t'$ such that $t' \neq \text{nil}$ and $(t', \varphi, t) \in \Delta$.
  - for every $(\text{nil}, \varphi, t) \in \Delta$, $s \models \varphi$.
  - for every $(t, \varphi, t') \in \Delta$, $s \models \varphi^o_{\alpha(t)}$.
  then $\Delta' = \Delta|_{\{t\}}, \alpha' = \alpha|_{\{t\}}, \Sigma' = \Sigma|_{\{t\}} \cup \{(\varphi, t') | (t, \varphi, t') \in \Delta$ and $t' \neq \text{nil}\}$
  $h' = h|_{\{t\}}$ and $s' = \gamma(s, \alpha(t))$.

A primitive task is chosen only if all its predecessors have been accomplished, its precondition is satisfied and the state constraints in $\Sigma$ except for those constraints ending with the primitive task, will hold after performing it. The ‘maintenance’ state constraints starting from the primitive task will be added into $\Sigma$.

- **Task decomposition:** for a compound task $t \in T$ and a method $(\alpha(t), (T_m, \Delta_m, \alpha_m)) \in M$, if $h(t) \cap T_m = \emptyset$, then $t \mapsto t'n$ and
  - $h' = h|_{T_m} \cup \{(l_m, h(t) \cup \{\alpha(t)\}) | t_m \in T_m\}$
  - $\Sigma' = \Sigma|_{T_m} \cup \{(\varphi, t) | (\varphi, t) \in \Sigma\}$

For a compound task, the method chosen to decomposed cannot contain an action which is its ancestor in order to avoid the recursion of tasks.

Note that the cardinality of $\Sigma$ may be exponential on size but $\text{Fml}(\Sigma)$ is polynomial which is restricted by the constraints in $M$. It entails that applying a step of acyclic progression is in $\mathcal{P}$.

If all tasks can be eliminated by the acyclic progression, then the TIHTNS problem has a solution:

**Lemma 3.** Given a TIHTNS problem $\mathcal{P}$, $\mathcal{P}$ has a solution if there is a sequence of acyclic progressions from $(s_1, t_{n1}, \emptyset, h_1)$ to $(s, t_{n0}, \emptyset, \emptyset)$ where $h_1 = \alpha_1$ and $t_{n0}$ is the empty task network.

As TIHTNS can cover lifted TIHTN whose plan-existence problem is 2-NEXPTIME-complete [Alford et al., 2015b], the plan-existence problem for TIHTNS is 2-NEXPTIME-hard. The next theorem states its completeness by proving the upper bound.

**Theorem 2.** Deciding whether a TIHTNS problem has a solution is 2-NEXPTIME-complete.

**Proof.** Theorem 1 reduces deciding the problem into checking all solutions generated by acyclic decomposition and Lemma 1 further reduces finding the problem into checking solutions with an upper bound on length. By introducing a counter on operators according to [Alford et al., 2015b], the length of a solution can be restricted with a polynomial translation on the problem. As the number of decomposing compound tasks is bounded double exponentially, it can ensure that every sequence of acyclic progression terminates after a double exponential number of steps of insertion, performing and decomposition. Therefore, by applying non-deterministically acyclic decomposition, either it can reach a solution or it cannot progress any more which entails that the problem has no solution. \hfill $\square$

**5 Conclusion**

In this paper, we extended TIHTN so that state constraints can be captured. We have shown that this extension does not increase the complexity while it can represent cover planning frameworks: lifted TIHTN and HR-HGN.

State constraints are written as linear temporal logic (LTL) formulas in PDDL3. It seems not difficult to adapt the notion of consistency with constraint to LTL formulas.

The experimental results in [Shivashankar et al., 2013] show that the completeness of methods can contribute to the performance of the HGN planners. It is also interesting to extend TIHTNS again with allowing method insertion. A promising reference is [Herzig et al., 2016] which provides a dynamic view of HGN planning and models decomposition in propositional dynamic logic.

**Acknowledgments**

We want to appreciate the reviewers for their insightful comments. Zhanhao Xiao is partially supported by CSC (Chinese Scholarship Council) and Hai Wan is partially supported by National Natural Science Foundation of China under grants 61573386, Natural Science Foundation of Guangdong Province under grant 2016A030313292 and Guangdong Province Science and Technology Plan projects under grant 2016B030305007. Laurent Perrussel is partially supported by ANR-11-LABX-0040-CIMI within the program ANR-11-IDEX-0002-02.
References


