Hierarchical Task Network Planning with Task Insertion and State Constraints

Zhanhao Xiao, Andreas Herzig, Laurent Perrussel, Hai Wan, Xiaoheng Su

Hierarchical Task Network Planning with Task Insertion and State Constraints

Zhanhao Xiao[1,2], **Andreas Herzig**[1,3], **Laurent Perrussel**[1], **Hai Wan**[4,5], and **Xiaoheng Su**[4]
[1]IRIT, University of Toulouse, Toulouse, France
[2]AIRG, Western Sydney University, Penrith, Australia
[3]IRIT, CNRS, Toulouse, France
[4]School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
[5]Guangdong 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-**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-**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-

composed into "clean the ground" (t_1) and "clean the table" (t_2). Suppose that "clean the ground" requires the full battery. Then there is no solution for the TIHTNS planning problem with such a state constraint.[1]

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,[2] noted $\mathcal{C}$. Each operator $o \in \mathcal{O}$ is represented as a triple of formulas: precondition $\mathsf{pre}(o)$, positive effect $\mathsf{add}(o)$, and negative effect $\mathsf{del}(o)$ where $\mathsf{add}(o), \mathsf{del}(o)$ are finite sets of atomic formulas. The variables in $\mathsf{add}(o)$ and $\mathsf{del}(o)$ should be associated with the variables in the operator so that they are instantiated simultaneously. For example, the positive effect of "openDoor(X)" being "Opened(X)", after grounding "openDoor(a)" has positive effect "Opened(a)" but not "Opened(b)". The positive and negative effect of an operator should not conflict, i.e., $\mathsf{add}(o) \cap \mathsf{del}(o) = \emptyset$ for every $o \in \mathcal{O}$. We suppose that $\mathcal{O}$ contains the 'empty' primitive operator skip where $\mathsf{pre}(\mathsf{skip}) = \top$ and $\mathsf{add}(\mathsf{skip}) = \mathsf{del}(\mathsf{skip}) = \emptyset$.

For a function $f : R \to S$, its restriction to a subset X of its domain is $f|_X = \{(r, s) \in f \mid r \in X\}$. Given a binary relation $Q \subseteq R \times R$, we define its restriction to $X \subseteq R$ by $Q|_X = Q \cap (X \times X)$; similarly for ternary relations. We extend the set union operator $\cup$ to tuples: $(Q_1, Q_2) \cup (Q_1', Q_2') = (Q_1 \cup Q_1', Q_2 \cup Q_2')$ and extend functions to sequences: $f(\langle t_1, \ldots, t_n \rangle) = \langle f(t_1), \ldots, f(t_n) \rangle$.

2.1 HTN Problems

Task networks A task network $\mathsf{tn} = (T, \Delta, \alpha)$ is a tuple, where

- T is a finite and non-empty set of task symbols;
- $\Delta \subseteq (T \cup \{\mathsf{nil}\}) \times \mathcal{L} \times (T \cup \{\mathsf{nil}\})$ is a set of constraints over T
- $\alpha : T \to \mathcal{C} \cup \mathcal{O}$ labels every task with an action.

With Function α, 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, φ, t_j) to denote a state constraint which intuitively means that formula φ must be true in all states between t_i and t_j. Specially, we introduce an idle task symbol nil which designates a task that is accomplished immediately: $(\mathsf{nil}, \varphi, t_j)$ and $(t_i, \varphi, \mathsf{nil})$ mean formula φ holds immediately before t_j and after t_i, respectively. We suppose nil only occurs in constraints. When φ is the truth constant $\top$ then the state constraint (t_i, φ, 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 task.

We say that two task networks $\mathsf{tn} = (T, \Delta, \alpha)$ and $\mathsf{tn}' = (T', \Delta', \alpha')$ are *isomorphic*, noted $\mathsf{tn} \cong \mathsf{tn}'$, if there exists a bijection $\delta : T \to T'$ 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 $\mathcal{D} = (\mathcal{L}, \mathcal{C}, \mathcal{O}, \mathcal{M})$ where $\mathcal{M}$ is a set of methods and $\mathcal{C} \cap \mathcal{O} = \emptyset$. An HTN problem is a tuple $\mathcal{P} = (\mathcal{D}, s_I, \mathsf{tn}_I)$ 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 $(\mathcal{L}, \mathcal{C}, \mathcal{O}, \mathcal{M}, s_I, \mathsf{tn}_I)$, we use $\mathcal{P} = (\mathcal{L}, C, O, M, s_I, \mathsf{tn}_I')$ 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 O determines a state-transition function $\gamma : 2^{\mathcal{L}_0} \times O \longrightarrow 2^{\mathcal{L}_0}$, where:

- $\gamma(s, o)$ is defined iff $s \models \mathsf{pre}(o)$;
- $\gamma(s, o) = (s \setminus \mathsf{del}(o)) \cup \mathsf{add}(o)$ if $\gamma(s, o)$ is defined.

[1] For the original TIHTN planning, there is an intuitive attempt to simulate that constraint: introducing a virtual primitive task pt whose precondition is "more than 10% battery" and introducing two ordering constraints $t_1 \prec pt$ and $pt \prec t_2$. Assume "charge" means to charge the battery, then the plan $\langle t_1$;"charge";pt;$t_2 \rangle$ is a solution of the original TIHTN planning problem. But it is counter-intuitive and the state constraint is still violated.

[2] In the original HTN, "non-primitive or compound task name" is used to distinguish from STRIPS-like actions, i.e., operators.

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 $\forall_{1 \leq i \leq n} \gamma(s_{i-1}, o_i) = s_i$.

Example 1. *Suppose we have an initial compound action $goMC$ 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* $\mathsf{At(Mc)}$*and* $\mathsf{At(Ma)}$ *to denote "being at Melbourne center" and "being at Melbourne airport". The primitive operators are:*

- $o_1 = (\top, \{\mathsf{At(Ma)}\}, \emptyset)$
- $o_2 = (\mathsf{At(Ma)}, \{\mathsf{At(Mc)}\}, \{\mathsf{At(Ma)}\})$

and the method is:

- $m = (goMC, \mathsf{tn}_m)$*, where* $\mathsf{tn}_m = (t_2, \emptyset, (t_2, o_1))$

Then the HTN problem $\mathcal{P}$ is $(\mathcal{L}, goMC, \{o_1, o_2\}, m, s_I, \mathsf{tn}_I)$ with $s_I = \emptyset$ and $\mathsf{tn}_I = (t_1, (t_1, \mathsf{At(Mc)}, \mathsf{nil}), (t_1, goMC))$.[3]

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 $*t$ and $t*$. As nil only occurs in constraints, we suppose the restriction of constraint set Δ to a set of tasks T is $\Delta|_T = \Delta \cap ((T \cup \{\mathsf{nil}\}) \times \mathcal{L} \times (T \cup \{\mathsf{nil}\}))$.

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

$$T' = (T \setminus \{t\}) \cup T_m \cup \{*t, t*\}$$

$$\begin{aligned}\Delta' = \Delta|_{T \setminus \{t\}} &\cup \Delta_m \cup \{(*t, \top, t_j), (t_j, \top, t*) \mid t_j \in T_m\} \\ &\cup \{(t_1, \varphi, *t) \mid (t_1, \varphi, t) \in \Delta\} \\ &\cup \{(t*, \varphi, t_2) \mid (t, \varphi, t_2) \in \Delta\}\end{aligned}$$

$$\alpha' = \alpha|_{T \setminus \{t\}} \cup \alpha_m \cup \{(*t, \mathsf{skip}), (t*, \mathsf{skip})\}$$

We write $\mathsf{tn} \xrightarrow[t,m]{} \mathsf{tn}'$ *when* 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 $*t$ and $t*$ in Δ'. More precisely, if φ holds before t then it also holds before $*t$ and if φ' holds after t then it also holds after $t*$. Subtasks should satisfy the inner constraints introduced by the decomposition method and should be performed between $*t$ and $t*$.

Example 2 (Example 1 continued)**.** *We apply the method m in tn_I to decompose t_1, i.e., $\mathsf{tn}_I \xrightarrow[t_1,m]{} \mathsf{tn}'$, where tn' is:*

$$\begin{aligned}T' &= \{t_2, *t_1, t_1*\} \\ \Delta' &= \{(t_2, \top, t_1*), (*t_1, \top, t_2),\ (t_1*, \mathsf{At(Mc)}, \mathsf{nil})\} \\ \alpha' &= \{(t_2, o_1), (*t_1, \mathsf{skip}), (t_1*, \mathsf{skip})\}\end{aligned}$$

[3] We will sometimes omit the braces of singleton sets.

Decomposition tree The hierarchical procedure of task decomposition can be viewed as a tree. Given an HTN domain $\mathcal{D} = (\mathcal{L}, C, O, M)$, a decomposition tree is a five-tuple $\mathsf{Tr} = (T, E, \Delta, \alpha, \beta)$ where (T, E) is a tree, rooted in t_0 which is a fresh task symbol, with nodes T and with directed edges E pointing towards the leaves; Δ is a set of constraints over T; function $\alpha : T \to C \cup O$ links tasks and actions where $\alpha(t_0) = c_0$ such that c_0 is a fresh compound action with an intuition "to achieve the initial task network". $\beta : T \to M'$ labels inner nodes with methods where $M' = M \cup \{(c_0, \mathsf{tn}_0)\}$ and tn_0 is some task network.

We write $\tau(\mathsf{Tr})$ for the nodes (tasks) of the decomposition tree Tr and $\mathsf{ch}(\mathsf{Tr}, t)$ for the direct children of $t \in \tau(\mathsf{Tr})$ in Tr. Hereafter $\mathsf{sub}(t)$ denotes the set of subtasks of t. The leaf nodes of Tr together with the constraints about these nodes define a task network, denoted by $\vartheta(\mathsf{Tr})$.

Definition 2 (valid decomposition tree)**.** *A decomposition tree* Tr *is valid w.r.t. an HTN problem $\mathcal{P} = (\mathcal{L}, C, O, M, s_I, \mathsf{tn}_I)$ iff the root node of* Tr *is t_0 where $\beta(t_0) = (c_0, \mathsf{tn}_I)$ and for any inner node t where $\beta(t) = (c, \mathsf{tn}_m)$, the followings hold:*

1. $\alpha(t) = c$

2. if $t = t_0$ then $\mathsf{ch}(\mathsf{Tr}, t) = \mathsf{sub}(t)$*, otherwise* $\mathsf{ch}(\mathsf{Tr}, t) = \mathsf{sub}(t) \cup \{*t, t*\}$ *such that*

- $(\mathsf{sub}(t), \Delta|_{\mathsf{sub}(t)}, \alpha|_{\mathsf{sub}(t)}) \cong \mathsf{tn}_m$
- *for every* $t' \in \mathsf{sub}(t)$*,* $(t', \top, t*), (*t, \top, t') \in \Delta$

3. for every $t' \in \tau(\mathsf{Tr}) \cup \{\mathsf{nil}\}$*:*

- *if* $(t, \varphi, t') \in \Delta$ *then* $(t*, \varphi, t') \in \Delta$
- *if* $(t', \varphi, t) \in \Delta$ *then* $(t', \varphi, *t) \in \Delta$

4. there are no constraints in Δ except for those demanded by criterion 2. and 3.

When tn' is reachable from tn by a finite sequence of decompositions then we write $\mathsf{tn} \longrightarrow_D^* \mathsf{tn}'$.

The decomposition tree focuses on the whole procedure of decomposition and records all intermediate tasks and constraints during the procedure. In contrast, in a task network the application of a decomposition method abandons the decomposed task. However, they are compatible as the following proposition states.

Proposition 1. *Given an HTN problem $\mathcal{P}$,* $\mathsf{tn}_I \longrightarrow_D^* \mathsf{tn}$ *iff there exists a valid decomposition tree* Tr *with respect to $\mathcal{P}$ where* $\vartheta(\mathsf{Tr}) = \mathsf{tn}$.

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 $\mathsf{tn} = (T, \Delta, \alpha)$ where $|T| = n$, let $\sigma : T \to \{1, \ldots, n\}$ be a bijection. We use σ to form a total ordering, noted $\sigma(\mathsf{tn})$, of tasks in T as: $\langle \sigma^-(1), \ldots, \sigma^-(n) \rangle$ where σ^- is the inverse function of σ, i.e., $\sigma^-(\sigma(t)) = t$. Suppose $\alpha(\sigma(\mathsf{tn}))$ 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 $\sigma(\mathsf{tn})$ is consistent with Δ in s_0 if for every $\sigma^-(i), \sigma^-(j) \in T$ the following hold:

- for every $(\mathsf{nil}, \varphi, \sigma^-(j)) \in \Delta$, $s_{j-1} \models \varphi$;

- for every $(\sigma^-(i), \varphi, \mathsf{nil}) \in \Delta$, $s_i \models \varphi$;
- for every $(\sigma^-(i), \varphi, \sigma^-(j)) \in \Delta$, $i < j$ and $s_k \models \varphi$ for every $i \leq k < j$.

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

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

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_I because apart from skip it only involves the operator o_1 and it is impossible to satisfy $\mathsf{At}(\mathsf{Mc})$. However, if we extend the task network by inserting some tasks, then we can make it executable.

Insertion Let $\mathsf{tn} = (T, \Delta, \alpha)$ and $\mathsf{tn}' = (T', \Delta', \alpha')$ be two task networks where tn' is primitive. Inserting tn' into tn results in the task network $\mathsf{tn}_1 = \mathsf{tn} \cup \mathsf{tn}'$.

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 $\mathsf{tn} \longrightarrow^*_{DI} \mathsf{tn}'$. The plan obtained only by decomposition 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_I. Given an HTN problem $\mathcal{P}$, we call σ_{tn,s_I} an HTN solution of $\mathcal{P}$ if $\mathsf{tn}_I \rightarrow^*_D \mathsf{tn}$; we call σ_{tn,s_I} a TIHTNS solution of $\mathcal{P}$ if $\mathsf{tn}_I \rightarrow^*_{DI} \mathsf{tn}$.

Example 3 (Example 2 continued)**.** *The plan $\langle *t_1, t_2, t_3, t_1* \rangle$ where $\alpha(t_3) = o_2$ is a TIHTNS solution of $\mathcal{P}$.*

When considering whether an HTN problem $\mathcal{P}$ has a TIHTNS solution, we call $\mathcal{P}$ 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 $\mathcal{P}$, suppose Tr is a valid decomposition tree with respect to $\mathcal{P}$ and tn' is the final primitive task network obtained from $\vartheta(\mathsf{Tr})$ by insertion and $\sigma_{\mathsf{tn}',s_I}$ is a solution of $\mathcal{P}$. Then all constraints in Tr are satisfied by $\sigma_{\mathsf{tn}',s_I}$.*

Sketch of proof. Suppose there are two compound tasks t_i and t_j in Tr and $\mathsf{tn}' = (T', \Delta', \alpha')$. For every $(t_i, \varphi, t_j) \in \Delta$, we have $(t_i*, \varphi, *t_j) \in \Delta'$ and then all states s_k such that $\sigma(t_i*) \leq k < \sigma(*t_j)$, $s_k \models \varphi$. For every $(t_i, \varphi, \mathsf{nil}) \in \Delta$, we have $(t_i*, \varphi, \mathsf{nil}), (st, \top, t_i*) \in \Delta'$ for all subtasks $st \in \mathsf{sub}(t_i)$ and then $s_{\sigma(t_i*)} \models \varphi$ and $\sigma(st) < \sigma(t_i*)$ for all $st \in \mathsf{sub}(t_i)$. The case of $(\mathsf{nil}, \varphi, t_j)$ is similar. If one or both of t_i, t_j are primitive then just consider $*t_j$ and t_i* as t_j and t_i, 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_2 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 $t_i \prec t_j$ in lifted TIHTN problem $\mathcal{P}$ with $(t_i, \top, t_j)$, we obtain a TIHTNS problem, noted $\Gamma_T(\mathcal{P})$. The following proposition shows that the two problems are equivalent.

Proposition 3. *For a lifted TIHTN problem $\mathcal{P}$, $\mathcal{P}$ has a solution iff $\Gamma_T(\mathcal{P})$ 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., $(t, \top, t')$, there will be $(t*, \top, t')$ introduced. As $(t_m, \top, t*)$ for all subtasks t_m are introduced, it entails that $(t_m, \top, t')$ 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 $\mathsf{gn} = (G, \prec)$ where G is a set of formulas in disjunctive normal form over ground literals, called goal formulas, and $\prec \subseteq G \times G$ is a

strict partial order on G. Similar with HTN problems, an HGN problem is a tuple $\mathcal{P} = (\mathcal{L}, O, M, s_I, \mathsf{gn}_I)$ where M is a set of HGN methods and gn_I is an initial goal network. For an HR-HGN problem, M is a 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 $|\mathsf{gn}_I|$, an HR-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 simulate a goal formula by requiring that the formula holds immediately before an empty task. Formally, given an HR-HGN problem $\mathcal{P} = (\mathcal{L}, O, s_I, \mathsf{gn}_I)$, we define a TIHTNS problem $\Gamma_G(\mathcal{P}) = (\mathcal{L}, \emptyset, O, \emptyset, s_I, \mathsf{tn})$ as:

- for every $g \in G_I$, $\lambda_g \in \mathsf{tn}$ and $(\mathsf{nil}, g, \lambda_g) \in \Delta$ and $\alpha(\lambda_g) = \mathsf{skip}$
- for every $g_i \prec g_j$, $(\lambda_{g_i}, \top, \lambda_{g_j})$

Next we show that the translation Γ_G is correct.

Proposition 4. *For an HGN problem $\mathcal{P}$, $\mathcal{P}$ has a solution iff $\Gamma_G(\mathcal{P})$ has a solution.*

Sketch of proof. Suppose $\sigma_{\mathsf{tn}', s_I}$ is a solution of $\Gamma_G(\mathcal{P})$. Then tn' is obtained from tn by an insertion. It forms a sequence $s_0, \ldots, s_n$ where $s_0 = s_I$. For every $g \in G_I$, we have $\lambda_g \in T'$ and $(\mathsf{nil}, g, \lambda_g) \in \Delta'$ then $s_i \models g$ where $\sigma(\lambda_g) = i$; for every $g_i \prec g_j$, we have $(\lambda_{g_i}, \top, \lambda_{g_j})\Delta'$ then $i' < j'$ and $s_{i'} \models g_i, s_{j'} \models g_j$,where $\sigma(\lambda_{g_i})$ and $\sigma(\lambda_{g_j})$. □

4 Complexity

In this section, we show that the solutions of TIHTNS can be obtained by acyclic decomposition and it does not increase the complexity, staying 2-**NEXPTIME**-complete.

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 $\mathsf{Tr}[t] = (T', E', \Delta|_{T'}, \alpha|_{T'}, \beta|_{T'})$ where (T', E') is the subtree in (T, E) which is rooted in t.

Let $\mathsf{Tr} = (T, E, \Delta, \alpha, \beta)$ be a decomposition tree and $t_i, t_j \in T$ be two nodes of Tr where t_i is an ancestor of t_j. We define the result of the subtree substitution on Tr that substitutes t_i by t_j, written $\mathsf{Tr}[t_i \leftarrow t_j] = (T', E', \Delta', \alpha|_{T'}, \beta|_{T'})$, where

$$
\begin{aligned}
T' &= (T \setminus \tau(\mathsf{Tr}[t_i])) \cup \tau(\mathsf{Tr}[t_j]) \\
E' &= E|_{T'} \cup \{(p, t_j) \mid (p, t_i) \in E\} \\
\Delta' &= (\Delta|_{T'} \setminus \{(\mathsf{nil}, \varphi_1, *t_j), (t_j*, \varphi_2, \mathsf{nil}) \in \Delta\}) \\
&\qquad \cup \{(t_j, \varphi_1, t_1), (t_j*, \varphi_1, t_1) \mid (t_i, \varphi_1, t_1) \in \Delta\} \\
&\qquad \cup \{(t_2, \varphi_2, t_j), (t_2, \varphi_2, *t_j) \mid (t_2, \varphi_2, t_i) \in \Delta\}
\end{aligned}
$$

Compared to [Geier and Bercher, 2011], the substitution operator here requires to remove t_j corresponding starting and terminating tasks before replacement because they are generated from t_j's parent which will be dropped. The following proposition states that the resulting tree is still valid.

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

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.

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

Proof. The right-to-left direction is straightforward. Now we prove the left-to-right direction.

Suppose $\sigma_{\mathsf{tn}, s_I}$ is a TIHTNS solution of $\mathcal{P}$ where $\mathsf{tn}_I \rightarrow_D^* \mathsf{tn}_1$ and tn is obtained from tn_1 by insertion. Then tn_1 is a primitive task network. If the sequence of decompositions from tn_I to tn_1 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 $\mathsf{Tr}[t_i \leftarrow t_j]$ is valid with respect to $\mathcal{P}$. By Proposition 1, there exists a primitive task network tn_2 such that $\mathsf{tn}_2 = \vartheta(\mathsf{Tr}[t_i \leftarrow t_j])$ and $\mathsf{tn}_I \rightarrow_D^* \mathsf{tn}_2$. As the substitution does not introduce any new node, the leaf nodes of $\mathsf{Tr}[t_i \leftarrow t_j]$ 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*, φ_1, t_1) and $(*t_j, \varphi_2, t_2)$. Now we replace all occurrences of t_j* with t_i*, $*t_j$ with $*t_i$ in $\mathsf{Tr}[t_i \leftarrow t_j]$. Then except for those constraints between subtasks of t_j and either t_j* or $*t_j$, in form of $(*t_j, \top, t)$ and $(t, \top, t_j*)$, the constraints in tn_2 are a subset of the constraints of tn_1. Because t_i is an ancestor of t_j in Tr, there must be some tasks $t', t'', \ldots, t^{(k)}$such that $(*t_i, \top, t'), (t', \top, t''), \ldots, (t^{(k)}, \top, *t_j)$. Thus, in the plan $\sigma_{\mathsf{tn}, s_I}$, $*t_i$ is before $*t_j$ and then before all subtasks of t_j. Then those constraints $(*t_j, \top, t)$ are satisfied; the case of t_j* is similar. So $\sigma_{\mathsf{tn}, s_I}$ includes all tasks in tn_2 and is consistent with all constraints in tn_2 and then tn can be obtained from tn_2 by inserting an appropriate task network. □

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

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

Sketch of proof. A valid decomposition tree generated by acyclic decomposition has at most $|T_I| \times (k+2)^{|C|}$ leaf nodes where $|C|$ is the number of ground compound actions that is $|\mathcal{C}| \times c^m$. As the number of states is bounded by $2^{|\mathcal{L}_0|}$ where $|\mathcal{L}_0|$ is the number of ground atoms that is $p \times c^n$, at most $2^{|\mathcal{L}_0|}$ tasks can be inserted between two neighbor tasks. □

As the state-transition function γ satisfies the frame axiom, before performing an operator we can check whether a formula is true in the next state. Given a formula φ, we use φ_o^- to denote the formula obtained from φ by replacing p with $\top$ and q with $\bot$ where $p \in \mathsf{add}(o)$ and $q \in \mathsf{del}(o)$. As all variables outside the effect of operator o keep their truth value, we can check whether a formula φ holds in the next state before operator o is performed, as the following lemma states:

Lemma 2. *$s \models \varphi_o^-$ iff $\gamma(s, o) \models \varphi$.*

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 Σ of pairs (φ, t) of formula and task which means φ should be satisfied until performing task t. We use $\mathsf{Fml}(\Sigma)$ to denote the conjunction of all formulas in Σ. 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 $h : T \longrightarrow 2^C$. Formally, we use a tuple $(s, \mathsf{tn}, \Sigma, h)$ to represent that task network tn still needs to be accomplished at the current state s. Given a tuple $(s, \mathsf{tn}, \Sigma, h)$, we define its acyclic progression is $(s', \mathsf{tn}', \Sigma', h')$ with $\mathsf{tn}' = (T', \Delta', \alpha')$, which is obtained from three possible options:

- Task insertion: if $s \models \mathsf{pre}(o) \wedge \mathsf{Fml}(\Sigma)_o^-$ then $s' = \gamma(s, o)$ and $\mathsf{tn}' = \mathsf{tn}, \Sigma' = \Sigma, h' = h$
  The task inserted should satisfy the state constraints.
- Task performing: for a primitive task $t \in T$, if $s \models \mathsf{pre}(\alpha(t)) \wedge \mathsf{Fml}(\Sigma_{-t})^-_{\alpha(t)}$ where $\Sigma_{-t} = \Sigma \cap (\mathcal{L} \times (T \setminus \{t\}))$ and the followings hold:
  - there is no t' such that $t' \neq \mathsf{nil}$ and $(t', \varphi, t) \in \Delta$
  - for every $(\mathsf{nil}, \varphi, t) \in \Delta$, $s \models \varphi$
  - for every $(t, \varphi, t') \in \Delta$, $s \models \varphi^-_{\alpha(t)}$

  then
  - $T' = T \setminus \{t\}$, $\Delta' = \Delta|_{T'}$, $\alpha' = \alpha|_{T'}$
  - $\Sigma' = \Sigma_{-t} \cup \{(\varphi, t') \mid (t, \varphi, t') \in \Delta \text{ and } t' \neq \mathsf{nil}\}$
  - $h' = h|_{T'}$ and $s' = \gamma(s, \alpha(t))$

  A primitive task is chosen only if its all predecessors have been accomplished, its precondition is satisfied and the state constraints in Σ, 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 Σ.
- 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 $\mathsf{tn} \xrightarrow[t,m]{} \mathsf{tn}'$ and
  - $h' = h|_{T'} \cup \{(t_m, h(t) \cup \{\alpha(t)\}) \mid t_m \in T_m\}$
  - $\Sigma' = \Sigma|_{T'} \cup \{(\varphi, *t) \mid (\varphi, t) \in \Sigma\}$

  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 Σ may be exponential on size but $\mathsf{Fml}(\Sigma)$ is polynomial which is restricted by the constraints in M. It entails that applying a step of acyclic progression is in **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 iff there is a sequence of acyclic progressions from $(s_I, \mathsf{tn}_I, \emptyset, h_I)$ to $(s, \mathsf{tn}_\emptyset, \emptyset, \emptyset)$ where $h_I = \alpha_I$ and $\mathsf{tn}_\emptyset$ 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 reduces further deciding 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 nondeterministically acyclic decomposition, either it can reach a solution or it cannot progress any more which entails that the problem has no solution. □

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 HTN planning and models decomposition in propositional dynamic logic.

Acknowledgments

We wants 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

[Alford *et al.*, 2015a] Ron Alford, Pascal Bercher, and David W Aha. Tight bounds for HTN planning. In *Proceedings of the 25th Intelligence Conference on Automated Planning and Scheduling (ICAPS-15)*, pages 7–15. Citeseer, 2015.

[Alford *et al.*, 2015b] Ron Alford, Pascal Bercher, and David W Aha. Tight bounds for HTN planning with task insertion. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, 2015.

[Alford *et al.*, 2016] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W. Aha. Hierarchical planning: Relating task and goal decomposition with task sharing. In *Proceedings of the 25fth International Joint Conference on Artificial Intelligence, (IJCAI-16)*, pages 3022–3029, 2016.

[Biundo and Schattenberg, 2001] Susanne Biundo and Bernd Schattenberg. From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In *Proceedings of the 6th European Conference on Planning (ECP-01)*, pages 157–168. AAAI Press, 2001.

[Biundo *et al.*, 2011] Susanne Biundo, Pascal Bercher, Thomas Geier, Felix Müller, and Bernd Schattenberg. Advanced user assistance based on AI planning. *Cognitive Systems Research*, 12(3):219–236, 2011.

[Erol *et al.*, 1994] Kutluhan Erol, James A. Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, volume 94, pages 1123–1128, 1994.

[Erol *et al.*, 1995] Kutluhan Erol, James A. Hendler, and Dana S. Nau. Semantics for hierarchical task-network planning. Technical report, DTIC Document, 1995.

[Geier and Bercher, 2011] Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-11)*, volume 22, pages 1955–1961, 2011.

[Gerevini and Long, 2005] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3. *Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, 75, 2005.

[Herzig *et al.*, 2016] Andreas Herzig, Laurent Perrussel, and Zhanhao Xiao. On hierarchical task networks. In *Proceedings of the 15th European Conference on Logics in Artificial Intelligence (JELIA)*, pages 551–557, 2016.

[Kambhampati *et al.*, 1998] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-98)*, pages 882–888, 1998.

[Lin *et al.*, 2008] Naiwen Lin, Ugur Kuter, and Evren Sirin. Web service composition with user preferences. In *Proceedings of European Semantic Web Conference (EWSC-08)*, pages 629–643. Springer, 2008.

[Shivashankar *et al.*, 2012] Vikas Shivashankar, Ugur Kuter, Dana Nau, and Ron Alford. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 981–988, 2012.

[Shivashankar *et al.*, 2013] Vikas Shivashankar, Ronald Alford, Ugur Kuter, and Dana S Nau. The godel planning system: A more perfect union of domain-independent and hierarchical planning. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-13)*, pages 2380–2386. AAAI Press, 2013.

[Shivashankar *et al.*, 2017] Vikas Shivashankar, Ron Alford, and David W. Aha. Incorporating domain-independent planning heuristics in hierarchical planning. In *Proceedings of the 31st Conference on Artificial Intelligence (AAAI)*, pages 3658–3664, 2017.

[Sohrabi and McIlraith, 2009] Shirin Sohrabi and Sheila A. McIlraith. Optimizing web service composition while enforcing regulations. In *Proceedings of the 8th International Semantic Web Conference, (ISWC),*, pages 601–617, 2009.

[Sohrabi *et al.*, 2009] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. HTN planning with preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1790–1797, 2009.