HTN Problem Spaces: Structure, Algorithms, Termination

Ron Alford
University of Maryland
College Park, MD, USA
ronwalf@cs.umd.edu

Vikas Shivashankar
University of Maryland
College Park, MD, USA
svikas@cs.umd.edu

Ugur Kuter
SIFT, LLC
Minneapolis, MN, USA
ukuter@sift.net

Dana Nau
University of Maryland
College Park, MD, USA
nau@cs.umd.edu

Abstract

For HTN planning, we formally characterize and classify four kinds of *problem spaces* in which each node represents a planning problem or subproblem. Two of the problem spaces are searched by current HTN planning algorithms; the other two problem spaces are new. This enables us to provide:

- Sufficient (and in one case, necessary) conditions for finiteness of each kind of problem space. The conditions can be evaluated up-front to see if an HTN planning problem is finite.
- Loop-detection tests that can be used in HTN planners to ensure termination when the problem space is finite.
- A way to compute the correct value for an upper-bound parameter in an HTN-to-PDDL translation algorithm published in IJCAI-2009.
- Planning algorithms that utilize the two new problem spaces to guarantee termination on broader classes of planning problems than previous HTN planning algorithms.

Introduction

Unlike HTN planning, classical planning is decidable (Erol, Nau, and Subrahmanian 1991). Thus it is possible to guarantee the termination of many classical planners, through the use of loop-checking tests to prevent the planner from generating infinite cyclic paths through a finite search space.

In contrast, HTN planning is only semi-decidable (Erol, Hendler, and Nau 1996), and every sound and complete HTN planner has a set of problems on which it will never return. Although some syntactic restrictions of HTN planning are fully decidable, efficient loop-detection tests have not yet been developed, and there is a gulf between the classes of HTN problems that are known to be decidable, and the classes of problems on which current HTN planners can guarantee termination.

Part of the reason for this gulf is that much less is understood about the search spaces of HTN planners than those of classical planners. Different kinds of HTN planners have different ways of combining plan generation with task decomposition—and the structure of the problem space can vary greatly depending on how those things are done.

Our contributions are as follows:

1. We characterize HTN planning as a search of a *problem space* in which each node is an HTN planning problem. We classify four different kinds of HTN problem spaces:
   - *Decomposition Space* (DS), which UMCP (Erol, Hendler, and Nau 1994) and the Landmark-Aware HTN Planner (Elkawkagy et al. 2011) search;
   - *Progression Space* (PS), which SHOP and SHOP2 (Nau et al. 1999; 2003) and HTNPBP (Sohrabi, Baier, and McIlraith 2009) search;
   - *Total Order Partition* decomposition and progression spaces (TODS and TOPS), two new kinds of problem spaces for which there are not yet any existing planners.
2. We provide sufficient (and in one case, necessary) conditions to guarantee that each kind of problem space will be finite. These conditions can be evaluated up-front to see if an HTN planning problem has a finite problem space. For each kind of problem space, we show that there are no broader conditions for finiteness that look only at possible decomposition of tasks.
3. We describe simple loop-detection tests that can be added to HTN planners that will guarantee termination when the problem space is finite.[1] These tests appear to be applicable to a wide variety of HTN planning problems, and their use will allow HTN planners to terminate in cases where they would not otherwise do so.
4. The HTN-to-PDDL translation algorithm in (Alford, Kuter, and Nau 2009) required a user-specified upper bound on the HTN recursion depth, and the translation was correct only when this bound was sufficiently high. By characterizing the translation as a mapping from HTN progression spaces into classical state spaces, we can compute a correct bound automatically whenever the finiteness conditions in Item 3 are satisfied.
5. We show that TODS and TOPS are finite for strictly broader classes of problems than DS and PS, and we provide new sound-and-complete HTN planning algorithms for TODS and TOPS. The algorithms are guaranteed to terminate whenever the problem space is finite.

[1]State-space classical planners often use a loop-detection test of the form "have we seen this state before?" In contrast, our loop-detection tests are basically "have we seen this problem before?"

HTN Planning

Here we present a set-theoretic HTN formalism, borrowing heavily from (Geier and Bercher 2011).

A task network is a partially-ordered multiset (POMSET) of task names. Given a set of task names X, a task network is a tuple $tn = (T, \prec, \alpha)$ such that:

- T is a finite nonempty set of symbols.
- $\prec$ is a partial order over T.
- $\alpha : T \to X$ is a map from symbols to a set of task names.

The symbols function as place holders for task names, allowing multiple instances of a task name to exist in the task network. We say a task network $(T, \prec, \alpha)$ is *equivalent* to another task network $(T', \prec', \alpha')$ if there is an isomorphism $\phi : T \to T'$ such that $\forall t_0, t_1 \in T \mid (t_0 \prec t_1) \Leftrightarrow (\phi(t_0) \prec' \phi(t_1))$ and $\forall t \in T \mid \alpha(t_0) = \alpha'(\phi(t_0))$. We refer to the set of all task networks over a set of task names X as TN_X.

An *HTN domain* is a tuple (S, C, O, M, γ), where:

- S is a finite set of states.
- C is a finite set of compound task names.
- O is a finite set of primitive task names, with $O \cap C = \emptyset$.
- $M \subset C \times TN_{C \cup O}$ is a set of methods over C and O.
- $\gamma : S \times O \to S$ is a partial function for state transitions. If $\gamma(s, a)$ is not defined, then a is not *applicable* in s.

We call a task network *primitive* if all the tasks map to a task name in O ($\forall t \in T.\alpha(t) \in O$). Otherwise, the task network is *non-primitive*.

We say a task name a_1 is *reachable* from a task name a_2 if there is a method $(a_2, tn) \in M$ where a_1 is a task name occurring in tn, or if there is a third task name a_3 such that a_1 is reachable from a_3 and a_3 is reachable from a_2.

Let $tn_1 = (T_1, \prec_1, \alpha_1)$ be a task network with a non-primitive task $t \in T_1$. If there is a method $m = (c, (T_m, \prec_m, \alpha_m)) \in M$ such that $c = \alpha(t)$, we can *decompose* t using m. Assume without loss of generality that $T_1 \cap T_m = \emptyset$. Then the decomposition of tn_1 by m into a task network tn_2 (written $tn_1 \xrightarrow{t,m}_D tn_2$) is given by:

$$\begin{array}{rcl}
T'_1 & := & T_1 \setminus \{t\} \\
T_2 & := & T'_1 \cup T_m \\
\prec'_1 & := & \{(t_1, t_2) \in \prec_1 \mid t_1 \neq t \wedge t_2 \neq t\} \\
\prec_2 & := & \prec'_1 \cup \prec_m \\
 & & \cup \{(t_1, t_2) \in T'_1 \times T_2 \mid (t_1, t) \in \prec_1\} \\
 & & \cup \{(t_2, t_1) \in T_2 \times T'_1 \mid (t, t_1) \in \prec_1\} \\
\alpha_2(t') & := & \begin{cases} \alpha_1(t') & \text{if } t' \in T'_1 \\ \alpha_m(t') & \text{if } t' \in T_m \end{cases} \\
tn_2 & := & (T_2, \prec_2, \alpha_2)
\end{array}$$

If there is a finite sequence of task decompositions from $tn_1 \to_D tn_2 \to_D \ldots \to_D tn_n$, then we write $tn_1 \to^*_D tn_n$.

An HTN problem is a tuple (D, s_0, tn), where $D = (S, C, O, M, \gamma)$ is an HTN domain, $s_0 \in S$ is a state in D, and $tn \in TN_{C \cup O}$ is a task network over the task names in D. A task network is *executable* in a state s_0 for domain D if tn is primitive and there exists some total ordering (consistent with $\prec$ in tn) over the tasks $t_1, \ldots, t_n$ and a list of additional states $s_1, \ldots, s_n$ such that

$$\forall_{i=0 \ldots n} \gamma(s_i, \alpha(t_{i+1})) = s_{i+1}$$

We refer to s_n as an *ending state*, and since $\prec$ is a partial order and γ is not guaranteed to be commutative, there may be many ending states for the executable task network with the same initial state.

We say an HTN problem (D, s_0, tn) is *solvable* if there is a sequence of decompositions $tn \to^*_D tn'$ such that tn' is executable in s_0.

Algorithm 1: DHTN(D, s_0, tn_0) A simple decomposition based HTN planner.

```
Input: $D = (S, C, O, M, \gamma)$ - an HTN domain
Input: $(s_0, tn_0)$ - an initial state and task network
$V \leftarrow Fringe \leftarrow \{tn_0\}$;
while $Fringe \neq \emptyset$ do
    Choose and remove some $tn \in Fringe$;
    if $tn$ is primitive and $(D, s_0, tn)$ is executable then
        return $tn$;
    $children \leftarrow \{tn' | tn \to_D tn'\}$;
    $Fringe \leftarrow Fringe \cup (children \setminus V)$;
    $V \leftarrow V \cup children$;
return fail;
```

Decomposition Problem Spaces

The definition of solvability leads to a natural definition of a problem space as a directed graph, where nodes are task networks, and edges are decompositions from one task network to another. The initial task network of a problem forms the root of the graph, and it is solvable if and only if there is a path in the graph from it to a primitive executable task network.

Formally, for an HTN domain $D = (S, C, O, M, \gamma)$, the directed graph (V, E) is the *decomposition problem space* of an HTN problem (D, s_0, tn_0) if and only if (V, E) is the minimal graph containing tn_0 such that $tn \in V$ and $tn \to_D tn'$ implies that $tn' \in V$ and $(tn, tn') \in E$.

Algorithm 1 (DHTN) shows a simple decomposition space HTN planner. DHTN starts off with the initial task network tn, and maintains a set of known decompositions V and a fringe of unexpanded decomposition.[2] At every iteration, it chooses some task network in its fringe. If tn is primitive and executable, DHTN returns tn. Otherwise, it removes tn from the fringe and adds tn's immediate decompositions. If at any time the fringe is exhausted, DHTN returns failure.

The computation $Fringe \leftarrow Fringe \cup (children \setminus V)$ guarantees that DHTN will never add a previously visited task network to the fringe. This is a simple loop-detection test that can be added to other HTN planning algorithms.

Given an HTN problem P, DHTN is sound, complete, and terminating for any problem for which the decomposition problem space is finite. However, if the problem space

[2] $Fringe$ and V functionally correspond to the open and closed sets of A*, respectively.

is infinite, DHTN's completeness depends on how it picks elements out of the fringe (with some acceptable choices being first-in-first-out or taking the network with the fewest tasks). If the problem has no solution and the problem space is infinite, then DHTN will not terminate no matter how elements are chosen from the fringe.

Relation to other work. The HTN decomposition problem space formalizes the spaces searched by existing planners such as UMCP (Erol, Hendler, and Nau 1994) and Elkawkagy et al.'s landmark-aware HTN planner. Unlike DHTN, neither UMCP nor Elkawkagy's planner check to see if they have already expanded a problem before, so a finite decomposition space is not enough to ensure their termination.

Decidability under decomposition

For an HTN domain $D = (S, C, O, M, \gamma)$, a *non-recursive* task name is one which has a finite k*-level-mapping*. k is a partial function from $C \cup O \to \mathbb{Z}^+$ defined via induction:

- For all $o \in O$, $k(o) = 0$.
- For $c \in C$, if there is a finite number $n \in \mathbb{Z}^+$ such that n is the greatest k-level of any subtask of c in M, then $k(c) = n + 1$.

If a task in a task network has a k-level, any decomposition of that task replaces it with a set of tasks which have a lower k-level. Since we can only repeat this a finite number of times, this leads to one of the first results of (Erol, Hendler, and Nau 1996):

Theorem 1 *Let $P = (D, s_0, tn_0)$ be an HTN planning problem with every task in tn_0 has a finite k-level mapping. Then the decomposition problem space for P is finite.*

So the decomposition problem space is finite for non-recursive problems, but limited recursion is fine as long as it does not increase the size of the task network. It turns out we can syntactically identify every problem for which the problem space is finite. To this end, we say that a task network tn is $\leq_1$*-stratifiable* if there exists a total preorder $\leq_1$ on the reachable task names of tn such that if c is a reachable task name of tn and $(c, (T, \prec, \alpha))$ is a method, then either:

- If the task network contains only one task ($T = \{t\}$), then the task name must not be on a higher stratum ($\alpha(t) \leq_1 c$)
- Otherwise, all task names must be on a lower stratum ($\forall_{t \in T} \alpha(t) <_1 c$).

Example 1 *Let $P = (D, s_0, tn_0)$ where tn_0 contains the single task r, $D = (S, C, O, M, \gamma)$, $C = \{r, s\}$, $O = \{a, b\}$, and:*

$$M = \begin{cases} (r, (\{x_1\}, \emptyset, \alpha(x_1) = s)) \\ (r, (\{x_1\}, \emptyset, \alpha(x_1) = a)) \\ (s, (\{x_1\}, \emptyset, \alpha(x_1) = r)) \\ (s, (\{x_1, x_2\}, \emptyset, \{\alpha(x_1) = b, \alpha(x_2) = b\})) \end{cases}$$

Then there is a two level $\leq_1$-stratification, where since r and s decompose to one another they must be on the same level, and a and b can be on the lower level together, since they are primitive.

Theorem 2 *Let $P(D, s_0, tn_0)$ be an HTN planning problem. Then the decomposition problem space for P is finite if and only if tn_0 is $\leq_1$-stratifiable.*

Proof. ($\Leftarrow$) If there exists a $\leq_1$-stratification, then every decomposition either produces a task network of the same size, or replaces a task with a set of tasks whose task names are in a lower level. Since the stratification is finite, the number of times repeated decomposition can increase the size of the task network is also finite.

($\Rightarrow$) If there is no $\leq_1$-stratification, then the transitive $\leq_1$ constraints are inconsistent. This means there exists a task name c and method $m = (c, tn_m)$ with more than one subtask such that $c \leq_1 \ldots <_1 c$, and so c is a reachable subtask of tn_m. Since c is a reachable subtask of tn_0, we can decompose tn_0 into a task network containing c, then repeatedly use the decomposition chain going through m to create a task network of arbitrary size. Since any finite problem space has a bound on the size of task networks appearing in it, the problem space for P is infinite. $\square$

Given an HTN domain D, we can find a $\leq_1$-stratification of a task network $tn = (T, \prec, \alpha)$ in time polynomial in D by performing a topological sort of the task names given the constraints. If this procedure fails, the decomposition space is infinite. If it succeeds, it returns a stratification. If the height of the stratification is h and the maximum number of tasks in a method is b, then the largest task network in the decomposition space of an HTN problem (D, s, tn) is bounded by $|T| \cdot b^h$.

Progression Problem Spaces

Decomposition spaces are not the only way to solve HTN planning problems. Let $P = (D, s, tn)$ be an HTN planning problem, where $D = (S, C, O, M, \gamma)$ and $tn = (T, \prec, \alpha)$. If there exists a task $t \in T$ with no predecessors (one such that $\forall_{t' \in T} t' \nprec t$) and its operator $\alpha(t) \in O$ is applicable in s, then we can *progress* the problem from (D, s, tn) to the problem $P' = (D, \gamma(s, \alpha(t)), tn \setminus \{t\})$ (where the notation $tn \setminus \{t\}$ simply means removing any occurrence of t from T, $\prec$, and α). If there exists an unconstrained task $t \in T$ which is non-primitive ($\alpha(t) \in C$), then any decomposition $tn \xrightarrow{t,m}_D tn'$ is a valid progression of (D, s, tn) to (D, s, tn'). We write progression as $P \xrightarrow{t}_P P'$.

Intuitively, a progression interleaves a decomposition with imposing an executable total order over the primitive tasks. The following theorem establishes an equivalence between these two paradigms:

Theorem 3 *An HTN problem P is solvable if and only if P is executable or there exists a P' such that $P \to_P P'$ and P' is solvable.*

Progression, then, also leads to a natural definition of the problem space as a directed graph. The *progression problem space* of an HTN problem (D, s_0, tn_0) is the minimal directed graph (V, E) containing (s_0, tn_0) such that $(s, tn) \in V$ and $(s, tn) \to_P (s', tn')$ implies that $(s', tn') \in V$ and $((s, tn), (s', tn')) \in E$.

Algorithm 2: PHTN(D, s_0, tn_0) A simple progression based HTN planner.

```
Input: $D = (S, C, O, M, \gamma)$ - an HTN domain
Input: $(s_0, tn_0)$ - an initial state and task network
$V \leftarrow Fringe \leftarrow \{(s_0, tn_0)\}$; $E \leftarrow \emptyset$;
while $Fringe \neq \emptyset$ do
    Choose some $(s, tn) \in Fringe$;
    if $tn$ is primitive and $(D, s, tn)$ is executable then
        return path in $(V, E)$ from $(s_0, tn_0)$ to $(s, tn)$;
    $children \leftarrow \{(s', tn') | (s, tn) \rightarrow_P (s', tn')\}$;
    $Fringe \leftarrow (Fringe \setminus \{(s, tn)\}) \cup (children \setminus V)$;
    $V \leftarrow V \cup children$;
    $E \leftarrow E \cup \{((s, tn), (s', tn')) | (s', tn') \in children\}$;
return fail;
```

PHTN (Algorithm 2) is a simple progression-based HTN planner. PHTN maintains a directed graph of HTN problems reachable from the initial problem and expands problems from the leaves of this graph. When it encounters a primitive executable problem, it returns the entire sequence of progressions from initial problem to its solution.

PHTN is sound, complete, and terminating for any HTN problem which has a finite progression problem space. The computation $Fringe \leftarrow (Fringe \setminus \{(s, tn)\}) \cup (children \setminus V)$ guarantees that PHTN will not add previously visited task networks to the fringe. This loop-detection test can be added to other HTN planning algorithms. But as with DHTN, if the problem space is infinite and there is no solution, PHTN will never return.

Relation to other work. Progression problem spaces provide an implicit formalization of the problem space behind several existing HTN planning works such as SHOP2 (Nau et al. 2003), HTNPBP (Sohrabi, Baier, and McIlraith 2009) and the HTN-PDDL translation in (Alford, Kuter, and Nau 2009). As with decomposition-based planners, finiteness does not guarantee termination; e.g., neither SHOP2 nor HTNPBP check if they have already expanded a problem.

Decidability under progression

Given an HTN domain D, suppose that the task network for every method in a domain consisted of a set of primitive tasks and at most one non-primitive task which is constrained to come after them. Erol, Hendler, and Nau call this a *regular* domain and prove a decidability result for regular HTN domains (Erol, Hendler, and Nau 1996). We adapt their result to our progression problem spaces as follows:

Theorem 4 *Given a regular HTN domain D, any HTN problem $P = (D, s_0, tn_0)$ has a finite progression problem space.*

The proof follows Erol, Hendler, and Nau's results. Intuitively, given any problem $P = (D, s_0, tn_0)$ with a regular domain, no progression can increase the number of non-primitive tasks in a task network. Furthermore, any primitive task introduced along with a non-primitive task must be progressed out of a task network before the non-primitive task can be decomposed. Thus, this bounds the size of the task networks in the progression problem space of P.

We can extend this class of decidable problems using the same stratification technique we used for decomposition. A task network tn is *$\leq_r$-stratifiable* if there exists a total preorder $\leq_r$ on the reachable task names of tn such that for every method $(c, (T, \prec, \alpha))$ with a reachable task name c:

- If there is a task $t_r \in T$ such that all other tasks are predecessors ($\forall_{t \in T, t \neq t_r} t \prec t_r$), then $\alpha(t_r) \leq_r c$. We call t_r the *last task* of $(T, \prec, \alpha)$.
- For all non-last tasks $t \in T$, $\alpha(t) <_r c$.

If an HTN planning problem P's task network is $\leq_r$-stratifiable, then any progression P replaces a task with at most one task of the same level, with the rest occurring at a lower level. Since the lower level tasks are constrained to come before this task, they must be progressed out of the task network before this task can be decomposed. Since the stratification is finite, this gives us a bound on the maximum size of the network, and produces our next finiteness result:

Theorem 5 *Given an HTN problem $P = (D, s_0, tn_0)$, if tn_0 is $\leq_r$-stratifiable, then the progression problem space of P is finite.*

What happens to the problem space if there is no stratification? Unlike with decomposition, now both the structure of the transition function and the set of methods can affect which problems are in the problem space. This limits our result on when the problem space must be infinite:

Theorem 6 *Given an HTN problem $P = (D, s_0, tn_0)$ such that every problem in the domain is solvable and there is no $\leq_r$-stratification of the reachable subtasks of tn_0, then the progression problem space of P is infinite.*

Proof. Given that there is no $\leq_r$-stratification, the $\leq_r$-constraints must be inconsistent, meaning there must be two reachable task names (not necessarily distinct) such that $b <_r c$ and $c \leq_r b$. So there is a method $m = (c, (T, \prec, \alpha))$, and two tasks $t_1, t_2 \in T$ such that $t_1 \not\prec t_2$ and $\alpha(t_2) = b$. Since c is a reachable task from tn_0 and every problem in P is solvable, there is a series of progressions such that c progresses to a task network containing itself. Since we did not need to progress t_1 out of tn_0 in order to expand t_2, we can use this loop to create a task network of arbitrary size. Thus the progression problem space of P is infinite. □

Since $\leq_r$-stratifiable is a strict broadening of the $\leq_1$-stratifiability definition, if a task network is $\leq_1$-stratifiable, it is also $\leq_r$-stratifiable. Like $\leq_1$-stratifications, we can find $\leq_r$-stratifications with a topological sort of the reachable task names of a task network. For an HTN problem P with the task network $tn = (T, \prec, \alpha)$ and a stratification of tn of height h and a largest method size of b, a task in a stratum can contribute at most 1 plus b times the bound of the strata below it to the maximum size of a task network reachable under progression. This gives a total bound of $|T| \cdot \sum_{i=0}^{h-1} b^i = |T| \cdot \frac{1-b^h}{1-b}$ for the maximum size of a task network in the progression problem space of P.[3]

[3]One can do much better than that by directly inspecting the stratification, but this is beyond the scope of the paper.

Relation to other work. The HTN to PDDL translation algorithm in (Alford, Kuter, and Nau 2009) essentially maps the progression problem space into a classical domain. The algorithm adds a fixed number of identifiers, specified by the user, to represent the structure of the task network. In order for the translation to be correct, there must be more identifiers available than there are tasks in the largest task network encountered under progression. If the initial task network is $\leq_r$-stratifiable, we can use the bound in the previous paragraph instead of asking the user.

Identifiability of finite progression spaces

Since we could identify the exact set of problems which have a finite decomposition space, there is a natural question about whether we can identify more problems which have a finite progression space.

Theorem 6 says that if the initial task network is not $\leq_r$-stratifiable, and every problem in the domain is be solvable, then the progression space is infinite. To say that every problem in the domain is solvable is equivalent to the following two conditions:

- Every operator is applicable in every state (i.e., $\forall_{s \in S, o \in O} \gamma(s, o)$ is defined).
- Every task has a primitive decomposition (transitively, not necessarily an immediate decomposition).

We can check the second condition recursively:

- Every primitive task has a primitive decomposition.
- Every task which has a method where all the tasks have a primitive decomposition also has a primitive decomposition.

Once all tasks with a primitive decomposition are marked, the rest have no primitive decomposition, and so are *trivially unsolvable*. Since any task network with a task that can't be decomposed into primitive network is unsolvable, we can preprocess the domain, removing any trivially unsolvable tasks and methods that refer to them.

This lets us identify every problem with a finite progression space that can be identified without looking at the state transition function:

Theorem 7 *Let P be an HTN planning problem and let P' be the HTN planning problem where all of the trivially unsolvable tasks of P are removed. If P' is not $\leq_r$-stratifiable, then the following holds:*

There does not exist a function $Q(P)$ which, without examining P's transition function, returns true if and only if P has a finite progression space.

Proof. Let $P = (D, s, tn)$ and $P' = (D', s, tn)$ be HTN problems, where D' is D without trivially unsolvable tasks, such that P has a finite progression space and P' has no $\leq_r$-stratification.

Let P_U and P'_U be P and P' where their domain's transition function has been replaced by γ_U, where $\forall_{s \in S, o \in O} \gamma_U(s, o) = s$. Then by Theorem 6, P'_U has an infinite progression space. Since adding methods and tasks cannot make a problem's infinite progression space finite, P_U also has an infinite progression space.

So if $Q(P)$ returns true and $Q(P_U)$ returns false, then Q must inspect the transition function of P and P_U, since that is their only difference. □

Total Order Partition Problem Spaces

This section describes two new problem spaces, based on our formulation of DS and PS. The new problem spaces will allow us to define new HTN planning algorithms which terminate for a broader class of HTN problems, as described in the subsequent section.

Total Order Partitions

Let $P = (D, s_0, tn)$ be an HTN planning problem where $tn = (T, \prec, \alpha)$ is its task network. Then the sequence of task networks $\langle (T_0, \prec_0, \alpha_0), \ldots, (T_k, \prec_k, \alpha_k) \rangle$ is a *total order partition* of tn if:

- Their union equals tn:
  - $T = \bigcup_{i=0}^{k} T_i$
  - $\prec = \bigcup_{i=0}^{k} \prec_i$
  - $\alpha = \bigcup_{i=0}^{k} \alpha_i$
- They are disjoint and tasks between networks are ordered, i.e. for any T_i and T_j such that $i < j$:
  - $T_i \cap T_j = \emptyset$
  - $\forall_{t_i \in T_i, t_j \in T_j} t_i \prec t_j$

A total order partition is a serialization of the task network into smaller problems, which we can attempt to solve the smaller problems sequentially without interactions from other tasks:

Theorem 8 *Let $P = (D, s_0, tn)$ and let $\langle tn_0, \ldots, tn_k \rangle$ be a total order partition of tn. Then P is solvable iff there exists a sequence of states $s_1, \ldots, s_{k+1}$ such that for all $i \leq k$ (D, s_i, tn_i) has a solution with an ending state of s_{i+1}.*

Proof Sketch. ($\Leftarrow$) If the sequence of partitions is solvable, then each tn_i decompose into some primitive task network tn'_i. Applying these decompositions to the corresponding tasks in tn gives you a primitive task network tn' which will have an ending state of s_{i+1}.

($\Rightarrow$) Suppose P has a primitive executable decomposition tn'. Since it is executable, there is a total order over the tasks of tn. Since decomposition preserves ordering, we can split that sequence into a solution for the partition. □

Given that $\prec$ is a consistent partial order, there will be a unique *longest total order partition* from which none of the reduced problems can be further reduced. If the longest total order partition contains only a single task newtork, we call that a *trivial partition*.

TOD and TOP problem spaces. Total order partitions give us two AND/OR problem spaces for HTN planning, one defined over decomposition and one over progression.

Given an HTN domain $D = (S, C, O, M, \gamma)$, the *total-order decomposition (TOD) problem space* for an HTN problem $P = (D, s_0, tn_0)$ is the minimal directed labeled graph (V, E) containing (s_0, tn_0) such that for every $(s, tn) \in V$:

- If tn has only a trivial total order partition and $tn \rightarrow_D tn'$, then (s, tn') is also in V with the edge $((s, tn), 0, (s, tn')) \in E$.
- Otherwise, let $\{tn_1, \ldots, tn_k\}$ be the longest total order partition of tn. Edges will point to reduced problems, labeled with where in the sequence the reduce problem lies. Then:
  - $(s, tn_1) \in V$ with the edge $((s, tn), 1, (s, tn_1))$.
  - For $i < k$, if there exists an edge $((s, tn), i, (s', tn_i)) \in E$ and (s', tn_i) has a solution with an ending state of s'', then there exists $(s'', tn_{i+1}) \in V$ with the edge $((s, tn), i + 1, (s'', tn_{i+1}))$.

For each problem $P = (s, tn)$ in the TOD problem space with a non-trivial total order partition $tn_1, \ldots, tn_k$, we label P's outgoing edges with the integer corresponding to its reduced task networked (edges corresponding to decomposition are given an arbitrary label of 0). If there is an edge $((s, tn), i, (s_i, tn_i)) \in E$ for $i > 1$, then by the definition there must also be a state s_{i-1} and an edge $((s, tn), i-1, (s_{i-1}, tn_{i-1})) \in E$ such that (s_{i-1}, tn_{i-1}) has a solution with an ending state of s_i. So if there is an edge $((s, tn), k, (s_k), tn_k)) \in E$ such that (D, s_k, tn_k) has a solution with an ending state of s_{k+1}, then there must be a chain of states that solves the partition, and so P has a solution with an ending state of s_{k+1}.

The *total-order progression (TOP) problem space* is defined similarly to the TOD problem space, replacing decomposition with progression.

Search in TOD and TOP spaces

We now describe two new HTN-planning algorithms, called TODHTN and TOPHTN, which perform an AND/OR search over the TOD and TOP problem spaces, respectively.

Algorithm 3 shows a high-level description of the TODHTN procedure. TODHTN maintains a set of variables as PHTN - a directed but now edge-labeled graph (V, E) of HTN problems, a set of HTN problems ($Fringe$), and a new map X, which maps HTN problems to a set of known possible end states. TODHTN then begins a two phase iterative process of *selecting* a node from the fringe to expand, then *propagating* the consequences through the graph.

Every iteration of TODHTN selects a problem (s, tn) from the fringe and examines its task network. Nothing is added to the graph in this phase, but instead TODHTN marks new edges and ending states to add later during the propagation phase. If the task network has a non-trivial total order partition $\langle tn_1, \ldots \rangle$, TODHTN marks the edge from (s, tn) to its first reduced child (s, tn_1). If the network is non-primitive, it marks the edges to all the immediate decompositions of tn. Otherwise the network is primitive, TODHTN marks all the possible ending states (if any).

The propagation phase itself is split into two parts: adding edges, and propagating ending states. When TODHTN adds an edge from one problem to its child, it checks to see if the child problem is already in the graph. If not, it adds the problem to the fringe. If the child is already in the current graph, TODHTN marks all of the child's known endings states for propagation to its parent (noting the edge label).

Algorithm 3: TODHTN(D, s, tn) A procedure to explore the TOD problem space.

```
Input: $D = (S, C, O, M, \gamma)$ - an HTN domain
Input: $(s, tn)$ - an initial state and task network
$V \leftarrow Fringe \leftarrow \{(s, tn)\}$;
$X(s, tn) \leftarrow \emptyset$;
while $Fringe \neq \emptyset$ & $X(s, tn) = \emptyset$ do
    // Pick and expand a fringe node
    Choose and remove some $(s', tn') \in Fringe$;
    if there is a total order $tn_1 \prec \ldots \prec tn_n$ over $tn'$ then
        Insert $((s', tn'), 1, (s', tn_1))$ into $NewE$;
    else if $tn'$ is nonprimitive then
        Insert $((s', tn'), 0, (s', tn''))$ into $NewE$ for every
        decomposition $tn' \rightarrow_D tn''$;
    else
        $(s', tn')$ is primitive, so add $((s', tn'), 0, s_e)$ to
        $NewX$ for every ending state $s_e$ of $(s', tn')$;

    while $NewE \neq \emptyset$ & $NewX \neq \emptyset$ do
        // Add edges, collect end states
        foreach $(v_1, k, v_2) \in NewE$ do
            if $v_2 \notin V$ then
                Insert $v_2$ into $Fringe$ and $V$;
            else
                For each $s_e \in X(v_2)$, add $(v_1, k, s_e)$ to
                $NewX$;
            Insert $(v_1, k, v_2)$ into $E$;
        $NewE \leftarrow \emptyset$;
        // Propagate end states
        while $NewX \neq \emptyset$ do
            Choose and remove some $((s_p, tn_p), k, s_e)$ from
            $NewX$;
            Let $tn_1, \ldots, tn_n$ be the longest total order
            partition over $tn_p$;
            if $0 < k < n$ then
                Insert $((s_p, tn_p), k + 1, (s_e, tn_{k+1}))$ into
                $NewE$;
            else if $s_e \notin X(s_p, tn_p)$ then
                Insert $s_e$ into $X(s_p, tn_p)$;
                foreach $(v, j, (s_p, tn_p)) \in E$ do
                    Insert $(v, j, s_e)$ into $NewX$;
if $X(s, tn) \neq \emptyset$ then
    return the preorder traversal of a subgraph of $(V, E)$
    showing a solution;
else
    return FAILURE;
```

When processing a new ending state s_e to propagate, if s_e is a solution to the interior part of a partition ($0 < k < n$), then that state is the start state for a next child in the partition, (s_e, tn_{k+1}), and TODHTN marks the edge to that problem for later addition. Otherwise, s_p is an end state for (s_p, tn_p), and if it is an ending state that TODHTN didn't already know about, it propagates it to the parents problems of (s_p, tn_p).

Since TODHTN follows the definitions of the TOD problem space in expanding nodes from the fringe, it is a sound HTN planner. If the TOD problem space is finite, then TODHTN is complete and will eventually terminate when it runs out of nodes from the fringe to expand and ending states to propagate. If the TOD problem space is infinite, then TODHTN's completeness depends upon how it chooses nodes out of the fringe (such as FIFO). If the problem is unsolvable and the problem space infinite, no matter how TODHTN chooses it will never return.

TOPHTN is defined nearly identically to TODHTN, substituting progression for decomposition.

Decidability under problem partition

We note that, since total order partitions split task networks into smaller task networks without introducing new tasks, the TOD and TOP problem spaces of a problem are finite if the decomposition or progression problem spaces are finite, respectively.

TOD and TOP are also finite for a strictly broader class of problems. Let $tn_1, \ldots, tn_k$ be the longest total-order partition of a task network tn for some number k. We say tn is $\leq_1$*-ordered*, if each tn_i, for $i = 1, \ldots, k$, in the longest total-order partition of tn is either a singleton or $\leq_1$-stratifiable. An HTN method (c, tn) is $\leq_1$*-ordered* if the task network tn is $\leq_1$-ordered. If every method in a domain is $\leq_1$-ordered, we call that domain $\leq_1$-ordered, and if an HTN planning problem's domain and initial task network is $\leq_1$-ordered, then so is the problem.

Theorem 9 *If P is $\leq_1$-ordered, it has a finite TOD problem space.*

Proof. Let $P = (D, s_0, tn_0)$ be a $\leq_1$-ordered HTN planning problem. Since the initial task network tn_0 is $\leq_1$-ordered by the condition of the theorem, it is either a singleton, $\leq_1$-stratifiable, or tn_0 has a non-trivial total order partition.

The proof proceeds by showing that every problem in TOD problem space of P has a task network tn, produced by decomposition over tn_0 in s_0, that satisfies at least one of the following conditions:

- *tn is a singleton.* Consider a node with a singleton task network. Its task is either primitive, i.e., the node has no children, or the node is non-primitive, i.e., by decomposition, its children each correspond to some method in D.
- *tn matches to the initial network (tn_0) or some HTN method's task network.* we have already shown the first case. For the second case, consider a non-singleton node (s, tn) with a task network tn that corresponds to some method in D, which means that tn can be produced by applying an HTN method to a nonprimitive task in a state. Then since that method is $\leq_1$-ordered, tn either has a non-trivial total order partition, or tn is $\leq_1$-stratifiable. In the first case, any children of (s, tn) have singleton task networks or are $\leq_1$-stratifiable.
- *tn is $\leq_1$-stratifiable:* Note that in this case, we already know there are only a finite number of problems reachable from (s, tn) in the TOD problem space.

Thus, since the number of states in D is finite, the TOD problem space of P is finite. □

We define $\leq_r$-ordered problems similarly, replacing $\leq_1$-stratification with $\leq_r$-stratification. The finiteness of the TOP problem space of $\leq_r$-ordered problems can be proved similarly.

We can prove an infiniteness theorem for TOD and TOP problem spaces which is similar to the progression space theorem. Here we pick the TOP space, since it provides a way to show the same for TOD as well:

Theorem 10 *Let $P = (D, s_0, tn_0)$ be an HTN problem where D has a method (c, tn) where c is a reachable task name of tn_0 and tn is not $\leq_r$-ordered. If every problem in D is solvable, then the TOP space of P is infinite.*

Proof. Since c is a reachable task name from tn_0 and every problem in D is solvable, we can reach some problem p using the method (c, tn). tn is not $\leq_r$-ordered, so its longest total order partition $\langle tn_1, \ldots, tn_k \rangle$ has some non-singleton task network tn_i (possibly equal to tn) which is not $\leq_r$-stratifiable.

From Theorem 6 we know that there is a chain of progressions that can produce a task network of arbitrary size in the progression space of any problem (D, s, tn_i). Let t_1 be the first progressed task in this chain. Since the partition of tn was maximal, there is some other task t_2 that is not constrained to come before or after t_1. Since the chain of progressions did not need to progress t_2 out of the task network, at no point in the sequence of progressions is there a non-trivial total order partition of a problem.

This means that the chain of progressions behaves identically in the TOP space as it does in the progression space, and so the TOP problem space of P is infinite. □

The proof of infiniteness for TOD spaces proceeds similarly, since once a task network has no non-trivial total order partition, no sequence of decompositions can restore it. As with progression spaces, $\leq_1$-ordered and $\leq_r$-ordered are the broadest class of finite problems identifiable without inspecting the state transition function.

Relation to other work. The $\leq_1$-ordered and $\leq_r$-ordered problems both include what (Erol, Hendler, and Nau 1996) calls *totally ordered problems.* A problem is *totally ordered* if there is a total order over the initial task network and over every method's task network. Where Erol, Hendler, and Nau prove that planning for totally ordered problem is decidable via a dynamic programming argument, we can repurpose the proof of Theorem 9 to provide a bound on the size of TOD and TOP spaces for totally ordered problems:

Theorem 11 *If P, where $D = (S, C, O, M, \alpha)$, is totally ordered, then there are at most $1 + |S| \cdot (|M| + |C| + |O|)$ vertices in both the TOD and TOP problem spaces of P.*

Computational complexity. Given a bound B on the number of vertices, TOD- and TOPHTN maintain $O\left(B \cdot (B + |S|)\right)$ space for the graph and map of vertices to ending states. Given that a vertex is only added to the fringe once and we only propagate end states from a given vertex $|S|$ times, TOD- and TOPHTN run in $O\left(B^2 \cdot |S|\right)$ time.

Practical Considerations

As a result of our theoretical analyses presented in this paper, a practical question arises from our work:

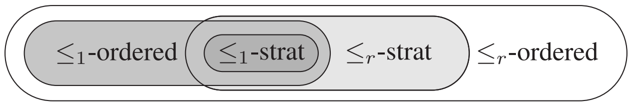

Figure 1: Every $\leq_1$-stratifiable problem is both $\leq_r$-stratifiable and $\leq_1$-ordered; every $\leq_r$-stratifiable problem is $\leq_r$-ordered; and every $\leq_1$-ordered problem problem is $\leq_r$-ordered.

Do existing HTN planning domains satisfy the finiteness criteria of HTN problem spaces?

To answer this question, we analyzed the properties of five different HTN domain models, namely Logistics, Blocks-World, Depots, Towers of Hanoi and Robot-Navigation, provided along with the SHOP2 distribution.[4] We showed that all of them are both $\leq_r$-stratifiable and $\leq_1$-ordered, with Logistics having a k-level mapping of depth 3. This suggests that typical HTN domains models (even complicated ones such as Blocks-World and Towers of Hanoi, which encode optimal problem-solving strategies) will most likely satisfy our finiteness criteria.

Thus, our theoretical and empirical analyses over HTN problem spaces suggest our polynomial-time computable conditions for the finiteness of the HTN problem spaces, and the loop-detection tests based on those finiteness conditions, will be practically useful in at least two ways:

- Authors of HTN domain descriptions will be able to use our theoretical finiteness conditions as guidelines so as to obtain guarantees on termination.
- HTN planning systems incorporating the search algorithms provided in this paper can determine whether conditions for finiteness are satisfied during planning. If the conditions are satisfied, the planner can freely choose any search procedure without worrying about termination, and therefore, completeness. Otherwise, the planner can choose to fall back onto a search strategy like breadth-first search that guarantees completeness. This is useful for systems such as SHOP2 where depth-first search is empirically much faster than breadth-first search.

Conclusions and Future Work

In this paper, we have provided a new formalization and classification of HTN problem spaces, that provides a better understanding of the conditions under which HTN planning algorithms can safely terminate (see Figure 1 for a summary). Although this work is primarily theoretical, we believe it may potentially lead to several practical benefits.

First, there is reason to believe that loop-checking tests based on our finiteness criteria will be widely applicable (see the previous section), and it should be straightforward to incorporate them into several existing HTN planners. We plan to do this in our future work. This will enable those planners to backtrack in cases where they otherwise might never return, thereby enabling the planners to solve a larger class of problems. It might also make some planners less sensitive to the order in which the HTN methods appear in the planner's input, making it easier to write HTN domain descriptions.

Second, our work provides a useful improvement to the HTN-to-PDDL translation algorithm in (Alford, Kuter, and Nau 2009). That algorithm requires a user-specified upper bound on the HTN recursion depth, and if the user supplies too low a bound, then the translation algorithm will produce a classical planning domain that is not a correct translation of the original HTN planning domain. By computing the correct bound automatically, we can make it easier to guarantee a correct translation.

Third, we have presented a new HTN planning algorithm that will terminate in cases where previous HTN planning algorithms would not terminate (not even with the incorporation of the loop-checking tests described above). In our future work, we hope to implement this algorithm and test its performance against existing HTN planners such as SHOP2 and Elkawkagy et al.'s Landmark-Aware HTN planner.

Acknowledgments. This work was supported in part by DARPA and U.S. Army Research Laboratory contract W911NF-11-C-0037, by Office of Naval Research grant N000141210430, and by a UMIACS New Research Frontiers Award. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited.

References

Alford, R.; Kuter, U.; and Nau, D. 2009. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *IJCAI*, 1629–1634.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2011. Landmark-aware strategies for hierarchical planning. In *HDIP 2011 3rd Workshop on Heuristics for Domain-independent Planning*, 73.

Erol, K.; Hendler, J.; and Nau, D. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, 249–254.

Erol, K.; Hendler, J.; and Nau, D. 1996. Complexity results for hierarchical task-network planning. *AMAI* 18.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1991. Complexity, decidability and undecidability results for domain-independent planning: A detailed analysis. *Artificial Intelligence* 76:75–88.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *IJCAI*, 1955–1961.

Nau, D.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *IJCAI*.

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

Sohrabi, S.; Baier, J.; and McIlraith, S. 2009. HTN planning with preferences. In *IJCAI*.

[4]http://www.cs.umd.edu/projects/shop/