Tight Bounds for HTN Planning with Task Insertion

Ron Alford\textsuperscript{1}  Pascal Bercher\textsuperscript{2}  David W. Aha\textsuperscript{3}

\textsuperscript{1}ASEE/NRL Postdoctoral Fellow  
ronald.alford.ctr@nrl.navy.mil

\textsuperscript{2}Ulm University  
pascal.bercher@uni-ulm.de

\textsuperscript{3}U.S. Naval Research Laboratory  
david.aha@nrl.navy.mil

July 31st, 2015
Theoretical applications of \(2^k\) Bottles of Beer on the Wall
Hierarchical Task Network (HTN) Planning

HTN planning is the problem of decomposing an initial task to accomplish into a sequence of executable steps.

- Encoding control knowledge for planning
  - Agent planning in robotics and simulation: ARTUE (2010), Johnny (2012)
  - Planning and mission generation for games: KILLZONE 2, Elder Scrolls, Armed Assault

- Representing and planning with processes and hierarchies
  - Composing web services (Sirin, 2004; Tang 2013; others)
  - Program configuration (Soltani, 2012)
The purpose of HTN planning is to complete a task. Tasks are either:

- **Primitive**, which corresponds to some concrete action we know how to perform, e.g., `walk(room, hall)`, or `drink(coffee)`
- **Non-primitive**, which is an abstract task. E.g. `travel(home, L.A.)`
- Must recursively decompose non-primitive tasks until we get primitive
tasks we know how to execute directly
- We are given a set of methods, which are recipes on how to accomplish abstract tasks. E.g., to travel from *home* to *L.A.*, we might decompose as follows:
Methods and Decomposition

- A method \((t, tn)\) is a non-primitive task \(t\) paired with a network \(tn\)

Method:

\[
\text{travel}(x, y) \quad \text{ticket}(a_1, a_2) \quad \text{fly}(a_1, a_2) \quad \text{travel}(a_2, y)
\]

Method:

\[
\text{travel}(x, y) \quad \text{taxi}(x, y)
\]

Method:

\[
\text{travel}(x, y) \quad \text{walk}(x, y)
\]

- We **decompose** a task network by replacing a node in the network with a corresponding method’s network.
Methods and Decomposition

- A method \((t, tn)\) is a non-primitive task \(t\) paired with a network \(tn\)

Method:

- **fly** \((?a_1, ?a_2)\)
- **ticket** \((?a_1, ?a_2)\)
- **travel** \((?x, ?a_1)\)
- **travel** \((?a_2, ?y)\)

We decompose a task network by replacing a node in the network with a corresponding method’s network.

Method:

- **travel** \((?x, ?y)\)
- **taxi** \((?x, ?y)\)
- **travel** \((?x, ?y)\)
- **walk** \((?x, ?y)\)
Methods and Decomposition

- A method \((t, tn)\) is a non-primitive task \(t\) paired with a network \(tn\)

We decompose a task network by replacing a node in the network with a corresponding method’s network.
Methods and Decomposition

- A method \((t, tn)\) is a non-primitive task \(t\) paired with a network \(tn\)

We decompose a task network by replacing a node in the network with a corresponding method’s network.

\[
\begin{align*}
\text{Method:} & \quad \text{fly}(a_1, a_2) \\
\text{Method:} & \quad \text{ticket}(a_1, a_2) \\
\text{Method:} & \quad \text{travel}(x, a_1) \\
\text{Method:} & \quad \text{travel}(a_2, y) \\
\text{Method:} & \quad \text{walk}(x, y) \\
\text{Method:} & \quad \text{taxi}(x, y) \\
\text{Method:} & \quad \text{travel}(x, y) \\
\text{Method:} & \quad \text{ticket}(DCA, LAX) \\
\text{Method:} & \quad \text{walk}(\text{home}, DCA) \\
\text{Method:} & \quad \text{fly}(DCA, LAX) \\
\text{Method:} & \quad \text{taxi}(LAX, L.A.)
\end{align*}
\]
Replanning with HTNs

```
Replanning with HTNs

ticket(DCA, LAX)  travel(home, DCA)  fly(DCA, LAX)  travel(LAX, L.A.)

travel(home, stop_1)  bus(stop_1, DCA)

walk(home, stop_1)  taxi(LAX, L.A.)
```
Replanning with HTNs

ticket(DCA, LAX)  travel(home, DCA)  fly(DCA, LAX)  travel(LAX, L.A.)

travel(home, stop_1)  bus(stop_1, DCA)

walk(home, stop_1)
Task Insertion

- An alternate set of semantics, HTN Planning with Task Insertion (TIHTN Planning) allows the insertion of tasks without a method.

- Developed by Kambhampati (1998) and Biundo (2002)
- Used as a model for replanning and search with partial HTN knowledge
Replanning with TIHTNs

\[ \text{ticket}(DCA, L.A.) \]
\[ \text{walk}(h, \text{stop}_1) \]
\[ \text{bus}(\text{stop}_1, DCA) \]

\[ \text{travel}(h, L.A.) \]
Replanning with TIHTNs

\[
\text{ticket}(DCA, L.A.)
\]

\[
\text{bus}(\text{stop}_1, DCA)
\]

\[
\text{walk}(h, \text{stop}_1)
\]

\[
\text{travel}(h, L.A.)
\]
Replanning with TIHTNs

ticket(DCA, L.A.)  travel(h, DCA)  ticket(DCA, SNA)  fly(DCA, SNA)  travel(SNA, L.A.)
bush(stop₁, DCA)  walk(h, stop₁)
Replanning with TIHTNs

diagram:

- ticket(DCA, L.A.)
- travel(h, DCA)
- ticket(DCA, SNA)
- fly(DCA, SNA)
- travel(SNA, L.A.)
- travel(h, stop₁)
- bus(stop₁, DCA)
- walk(h, stop₁)
Replanning with TIHTNs

- ticket(DCA, L.A.)
- travel(h, DCA)
- ticket(DCA, SNA)
- fly(DCA, SNA)
- travel(SNA, L.A.)
- travel(h, stop₁)
- bus(stop₁, DCA)
- walk(h, stop₁)
Replanning with TIHTNs

ticket(DCA, L.A.)

travel(h, DCA)

ticket(DCA, SNA)

fly(DCA, SNA)

travel(SNA, L.A.)

travel(h, stop₁)

bus(stop₁, DCA)

walk(h, stop₁)

return_ticket
Replanning with TIHTNs
Why do we study complexity?

- TIHTN is decidable (Geier & Bercher, 2011)
  - Very little research on algorithms for TIHTN planning
  - No theoretical comparison, none guaranteed to terminate
- Decision procedures are important for model checking, non-deterministic planning, translation-based planning
- Method structures impact decidability and complexity:
  - Whether the method’s tasks are totally-ordered
  - Where in the method recursion may occur
  - Whether the domain is relational (non-propositional)
Why do we study complexity?

- TIHTN is decidable (Geier & Bercher, 2011)
  - Very little research on algorithms for TIHTN planning
  - No theoretical comparison, none guaranteed to terminate

- Decision procedures are important for model checking, non-deterministic planning, translation-based planning

- Method structures impact decidability and complexity:
  - Whether the method’s tasks are totally-ordered
  - Where in the method recursion may occur
  - Whether the domain is relational (non-propositional)
TIHTN planning is decidable (Geier & Bercher, 2011)

If a TIHTN problem is solvable, there is a solution with an acyclic decomposition
TIHTN planning is decidable (Geier & Bercher, 2011)

If a TIHTN problem is solvable, there is a solution with an acyclic decomposition.
Planning with finite trees
Acyclic HTN planning

Acyclic HTNs:
- No recursion allowed
- Useful for program configuration (Soltani, 2012)

Method (totally-ordered):
- work → drink → write

Method (partially-ordered):
- work more → work
- work more → work

Method (partially-ordered):
- workaholic → work more → work more
Acyclic HTN planning, with and without task insertion

Acyclic HTNs:
- No recursion allowed
- $\text{NEXPTIME}$-complete
- Every decomposition in $\text{NEXPTIME}$ proof has same number of tasks

With insertion:
- Use counting technique to limit insertions
- TIHTN planning is $\text{NEXPTIME}$-complete

Method (totally-ordered):

```
work -> drink -> write
```

Method (partially-ordered):

```
work_more -> work
```

```
work_more -> work
```

```
work_more -> work_more
```

Method (partially-ordered):

```
workaholic
```

```
work_more
```

```
work_more
```

```
work_more
```
Totally-Ordered Propositional HTN Complexity

Standard Semantics (ICAPS’15)

EXPTIME

PSPACE

NP

Task-Insertion Semantics (new)

Arbitrary

Tail-Recursive

Regular

Acyclic

Regular-Acyclic

Arbitrary

Tail-Recursive

Regular

Acyclic

Regular-Acyclic
Partially-Ordered Propositional HTN Complexity

Standard Semantics (ICAPS’15)  

- **semidecidable**  
  - **Arbitrary**  
  - **Tail-Recursive**  
  - **Acyclic**

- **EXPSPACE**  

- **NEXPTIME**

- **EXPTIME**  

- **PSPACE**  

- **NP**

Task-Insertion Semantics (new)  

- **Arbitrary**  
  - **Tail-Recursive**  
  - **Acyclic**

- **Regular**  

- **Regular-Acyclic**

Alford, Bercher, and Aha

Bounds for TIHTNs

July 31st, 2015
Contributions

Provided:
- New complexity bounds for all combinations of:
  - Propositional or non-propositional methods and actions
  - Partially or totally-ordered methods
  - General or regular method structures
- Theoretically-efficient algorithm for TIHTN planning
Implications

For end users:

Totally-ordered propositional TIHTNs are PSPACE-complete. Poly-time (quadratic) translatable to propositional STRIPS. Probably only useful when it is linear.

For search people: We state-space search algorithm for TIHTN planning. $A^*$ needs duplicate detection.

DAG isomorphism is GI-complete. Totally-ordered problems can use lists. Acyclic progression is just one blocking technique (may be others).

For domain-independent heuristicists: Very few domain-independent TIHTN heuristics. Delete-free TIHTN planning is poly-time decidable, but?
Implications

For end users: Wait five years?
Implications

For end users: Wait five years?
- Totally-ordered propositional TIHTN\textregistered{}s are PSPACE-complete
  - Poly-time (quadratic) translatable to propositional STRIPS
  - Probably only useful when it is linear
Implications

For end users: Wait five years?
- Totally-ordered propositional TIHTNs are PSPACE-complete
  - Poly-time (quadratic) translatable to propositional STRIPS
  - Probably only useful when it is linear

For search people: We state-space search algorithm for TIHTN planning
- $A^*$ needs duplicate detection
  - DAG isomorphism is GI-complete
  - Totally-ordered problems can use lists
- Acyclic progression is just one blocking technique (may be others)
Implications

For end users: Wait five years?
- Totally-ordered propositional TIHTNs are PSPACE-complete
  - Poly-time (quadratic) translatable to propositional STRIPS
  - Probably only useful when it is linear

For search people: We state-space search algorithm for TIHTN planning
- $A^*$ needs duplicate detection
  - DAG isomorphism is GI-complete
  - Totally-ordered problems can use lists
- Acyclic progression is just one blocking technique (may be others)

For domain-independent heuristicists:
- Very few domain-independent TIHTN heuristics
- Delete-free TIHTN planning is poly-time decidable, but?
Method structures: Regular & Tail-Recursive

- Only allowed to recurse through last task
- Only last task in regular HTNs can be non-primitive
- Tail-recursion generalizes acyclic HTNs

Method (totally-ordered regular):

```
work → drink → write → work
```

Method (partially-ordered regular):

```
work → drink
work → write
```

Method (totally-ordered tail-recursive):

```
live → work → sleep → live
```
Method structures: Arbitrary

Partially-ordered arbitrary problems:
- Can express intersection of CFGs
- Semi-decidable

Totally-ordered arbitrary problems:
- Sometimes used for program representation
  - Web services (Sirin, 2004)
  - Java security (Kuter, 2015)
- Relatively easy to encode 2-player game tree search (chess, checkers)

Method (partially-ordered):
- live
- work
- sleep

Method (totally-ordered):
- live
- work
- brew
- drink
Task networks are size-bounded under acyclic progression.

Size bounds if $n$ is the number of task names and $m$ is the size of the largest method:
- $m \cdot n$ for totally-ordered problems
- $m^n$ for partially-ordered problems