Abstract

HTN planning is the problem of decomposing an initial task to accomplish into a sequence of executable steps. HTN planning with Task Insertion (TIHTN planning) allows the insertion of operators from outside the method hierarchy, which:

- Hybridizes classical planning with HTN planning.
- Allows partial task hierarchies with "missing required tasks" inserted by planner.
- We provide tight complexity bounds for TIHTN planning along two axis: whether variables are allowed and whether methods must be totally ordered.

HTN Planning (Overview)

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:

\[
\text{travel(h, L.A.)} \\
\text{my tickets} \Rightarrow \text{travel(h, a1)} \\
\text{fly(a1, a2)} \Rightarrow \text{travel(a2, L.A.)}
\]

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.
- An alternate set of semantics, HTN Planning with Task Insertion (TIHTN Planning) allows the insertion of tasks without a method.
- To a great extent, we can characterize the complexity of HTN and TIHTN planning by the structure of a problem’s methods: whether the methods are totally grounded, whether the methods are totally ordered, and where in the method recursion occurs.

Method structures: Acyclic

Method (totally-ordered):

\[
\text{work} \Rightarrow \text{drink} \Rightarrow \text{write} \Rightarrow \text{sleep}
\]

Method (partially-ordered):

\[
\text{work} \Rightarrow \text{drink} \Rightarrow \text{write} \Rightarrow \text{travel}
\]

Method structures: Regular

Method (totally-ordered):

\[
\text{work} \Rightarrow \text{drink} \Rightarrow \text{write} \Rightarrow \text{sleep}
\]

Method (partially-ordered):

\[
\text{work} \Rightarrow \text{drink} \Rightarrow \text{write} \Rightarrow \text{travel}
\]

Method structures: Tail Recursive

Method (totally-ordered):

\[
\text{live} \Rightarrow \text{work} \Rightarrow \text{sleep} \Rightarrow \text{live}
\]

Method (partially-ordered):

\[
\text{live} \Rightarrow \text{work} \Rightarrow \text{sleep} \Rightarrow \text{live}
\]

Method structures: Arbitrary

Method (totally-ordered):

\[
\text{work} \Rightarrow \text{brew} \Rightarrow \text{work} \Rightarrow \text{drink}
\]

Method (partially-ordered):

\[
\text{live} \Rightarrow \text{work} \Rightarrow \text{sleep} \Rightarrow \text{live}
\]

Conclusions

- Totally-ordered TIHTN planning has the same worst-case complexity as classical planning.
- Partially-ordered TIHTN planning has the same worst-case complexity as partially-ordered acyclic HTN planning (NEXPTIME), and is sometimes simpler.
- Future Work: In the paper, we provide a new planning technique for TIHTN planning, called acyclic progression, that let us define worst-case efficient TIHTN planning algorithms. Theoretical efficiency is not implementation efficiency, and so we hope to implement and evaluate acyclic progression.