Lifted Tree Path Planner

Gaspard Quenard, Damien Pellier, Humbert Fiorino

Lifted Tree Path Planner

Gaspard Quenard, Damien Pellier, Humbert Fiorino
Univ. Grenoble Alpes - LIG
F-38000 Grenoble, France
{firstname.name}@univ-grenoble-alpes.fr

Abstract

In this paper, we present our planner named LTP which stand for Lifted Tree Path aimed at solving Totally Ordered Hierarchical Task Network (TOHTN) problems. Our planner is based on the Satisfiability (SAT) planning paradigm and builds upon the concepts of the Lilotane planner (Schreiber (2021)), which has scored 2nd in the last IPC in the HTN Total Order track.

Introduction

Satisfiability (SAT) planning is a widely-used planning paradigm that employs Boolean satisfiability solvers to find solutions for planning problems (Kautz et al. (1992, 2006)). SAT solvers are efficient tools for solving propositional logic problems. The main challenge in SAT planning lies in identifying and formulating the appropriate set of rules and constraints that effectively encode a given planning problem into SAT clauses. Once the planning problem is encoded into SAT clauses, the solution process relies on the underlying SAT solver to efficiently search for satisfying assignments.

Several SAT planners have been developed to encode TOHTN problems (Behnke et al. (2018); Schreiber et al. (2019); Schreiber (2021)). These planners utilize a structure referred to as a hierarchical tree to represent the problem hierarchy up to a certain depth. This hierarchical tree is subsequently used to encode the set of relevant SAT clauses.

The difference between previous approaches and LTP (Lifted Task Planning) is that the latter does not directly encode the entire hierarchy of the problem into propositional logic. Instead, it selectively extracts only the primitives from the hierarchical tree that may appear in valid plans and encodes them into propositional logic. It focuses solely on the actions of the plan rather than the full hierarchy. Therefore, LTP does not utilize boolean variables to encode tasks or methods during the SAT clause encoding process.

Lifted HTN Planning Problem

In this section, we formally define a lifted HTN planning problem. Our formalism is based on a quantifier-free first-order predicate logic $\mathcal{L} = (P, T, V, C)$ inspired from Höller et al. (2020). All sets mentioned in the following are finite. T is a set of *type symbols* and V a set of typed variable symbols. P is a set of *predicate symbols*, each having a arity. The arity defines its number of parameter variables (taken from V). C is a set of typed constants, syntactic representations of the objects in the real world.

Task, Task Networks, Action and Methods

The two key concepts in HTN Planning are the concept of task and task network.

A task is given by a name and a list of parameters. There are two kinds of tasks: the primitive task and the abstract tasks. Unlike primitive tasks that do change the state of the world, *abstract tasks* do not. They are names referring to other tasks (primitive or abstract) that must be achieved with respect to some constraints.

The basic data structure in HTN planning is a *task network*, which represents a partially ordered multi-set of tasks.

Definition 1 (Task Network) *A task network w over a set of* task names *T (first-order atoms) is a tuple $(I, \alpha, \prec)$ with the following elements: I is a (possibly empty) set of* task identifiers*, $\alpha : I \rightarrow X$ maps task identifiers to task names and $\prec$ is a strict partial order over I.*

Task identifiers are arbitrary symbols which serve as place holders for the actual tasks they represent. We call a task network *ground* if all parameters are bound to (or replaced by) constants from C and *primitive* if all its tasks are primitive.

Primitive and abstract tasks in a task network can be achieved respectively by applying *actions* and *methods* defined below.

An *action* a is a tuple $(\textit{name}(a), \textit{precond}(a), \textit{effect}(a))$. $\textit{name}(a)$ is its *task name*, a first-order atom such as $drive(?from, ?to)$ consisting of the (actual) name followed by a list of typed parameter variables. $\textit{precond}(a)$ is its *precondition*, a first-order formula over literals over $\mathcal{L}$'s predicates. $\textit{effect}(a)$ is its *effect*, a conjunction of literals over $\mathcal{L}$'s predicates (often divided into the positive $\textit{effect}^+$ and the negative effects $\textit{effect}^-$). All variables used in $\textit{precond}(a)$ and $\textit{effect}(a)$ have to be parameters of $\textit{name}(a)$. We require that action names $\textit{name}(a)$ are unique.

A *method* m is a tuple (t, w) of a compound task name t and a task network w. A method express how the task t can be decomposed into sub-tasks defined in w to be achieved.

Definition 2 (Planning Problem) *A* planning problem *P is a tuple $(L, T, A, M, s_0, w_0, g)$, where: $\mathcal{L}$ is the underlying predicate logic, T is the set of primitive and compound tasks,*

A is a set of actions, M is a set of decomposition methods, s_0 is the initial state, a ground conjunction of positive literals over the predicates, w_0 is the initial task network (not necessarily ground) and g is the goal description, a first-order formula over the predicates (not necessarily ground).

HTN Planning Solutions

Solutions in HTN planning are executable, ground, primitive task networks that can be obtained from the problem's initial task network via applying methods, adding ordering constraints, and grounding. Lifted problems are a compact representation of their ground instantiations. We define the semantics of a lifted problem (i.e., the set of solutions) in terms of the standard semantics of its ground instantiation.

Definition 3 (HTN planning problem) *A* HTN planning problem *P is a tuple $(L, T, A, M, w_0, s_0, g)$, where L is a finite set of ground atoms, T is a set of task names (primitive and abstract), A is a set of actions and M is the set of methods, w_0 is the initial task network to decompose and s_0 and g to subset of ground atom of L that represent respectively the initial state and the goal to achieve.*

An action $a \in A$ is called executable in a state s if and only if $s \subset \mathit{precond}(a)$. The state transition function $\gamma : S \times A \to S$ is defined as follows: If a is executable in s, then $\gamma(s, a) = (s \setminus \mathit{effect}^-(a)) \cup \mathit{effect}^+(a)$, otherwise $\gamma(s, a)$ is undefined. The extension of γ to action sequences, $\gamma^* : S \times A^* \to S$ is defined straightforwardly.

It remains to define how to decompose a task network into a task network containing only primitive tasks by using methods. Let a task network $w_1 = (I_1, \alpha_1, \prec_1)$ and a task t such that $t = \alpha_1(i)$. Consider a method m such that $m = (t, w_m)$ and $w_m = (I_m, \alpha_m, \prec_m)$ that decomposes t. Then, m refines w_1 into $w_2 = (I_2, \alpha_2, \prec_2)$ as follow:

$$
\begin{aligned}
I_2 &= (I_1 - \{i\}) \cup I_m \\
\alpha_2 &= (\alpha_1 \cup \alpha_m) - \{(i, t)\} \\
\prec_2 &= \prec_1 \cup \prec_m \cup \{(i_1, i_2) \in I_1 \times I_m \mid (i_1, t) \in \prec_1\} \\
&\quad \cup \{(i_1, i_2)\} \in I_m \times I_1 \mid (t, i_2) \in \prec_1\}) \\
&\quad - \{i', i'') \in I_1 \times I_1 \mid i' = t \text{ or } i'' = t\}
\end{aligned}
$$

Now we can formally define the what is a task network solution.

Definition 4 (Solutions) *Let $P = (L, T, A, M, w_0, s_0, g)$ be a planning problem and $w_s = (I_S, \alpha_s, \prec_s)$ a task network. w_s is a* solution to an HTN planning problem *P if and only if*

- *w_s contains only primitive tasks;*
- *There is a sequence of decompositions from w_0 to w_s;*
- *There is a linearization π of the task identifiers of $I_s = i_1, \ldots, i_n$ with $n = |I_s|$, such that $\pi = \alpha_s(i_1), \ldots, \alpha_s(i_n)$ executable in s_0 and such that $\gamma(s_0, \pi) \subseteq g$.*

Hierarchical Tree

LTP utilizes the same hierarchical tree structure as the lilotane and TreeRex planners.

The hierarchical tree can be described as a sequence of hierarchical layers, where each layer is an array of positions, each containing a set of elements. These elements can be facts, reductions, or actions. The layers are computed incrementally, starting with an initial layer (L0) that includes the initial reduction. Subsequently, each layer is defined by including all operations that match a subtask of some operation from the previous layer.

Figure 1 illustrates an example of a hierarchical tree containing three layers for a problem in the Transport domain, as defined in the Lilotane paper. In this example, Lilotane encodes the entire decomposition tree into SAT clauses. However, LTP differs by keeping only the last layer of the decomposition tree. From this layer, it encodes only the actions that may be part of a solution plan as illustrated in the figure 2. The ordered constraints between these actions can be inferred from the hierarchical tree, and the method's preconditions can be encoded to the relevant actions in their first subtask.

Instantiation

The general planning procedure of LTP is similar to the other SAT planners for TOHTN problems:

1. Initialize the first layer (l0) of the hierarchial tree following the problem description.
2. Construct the next layer (l+1) of the hierarchical tree on the basis of the layer l.
3. Use the current hierarchical tree to encode the SAT clauses.
4. Launch the solver. If no solution is found, goto 2

References

Dominik Schreiber. Lilotane: A lifted sat-based approach to hierarchical planning. *Journal of artificial intelligence research*, 70:1117–1181, 2021.

Henry A Kautz, Bart Selman, et al. Planning as satisfiability. In *ECAI*, volume 92, pages 359–363. Citeseer, 1992.

Henry Kautz, Bart Selman, and Joerg Hoffmann. Satplan: Planning as satisfiability. In *5th international planning competition*, volume 20, page 156, 2006.

Gregor Behnke, Daniel Höller, and Susanne Biundo. totsat-totally-ordered hierarchical planning through sat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Dominik Schreiber, Damien Pellier, Humbert Fiorino, et al. Tree-rex: Sat-based tree exploration for efficient and high-quality htn planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 382–390, 2019.

Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. HDDL: an extension to PDDL for expressing hierarchical planning problems. In *The AAAI Conference on Artificial Intelligence*, pages 9883–9891, 2020.

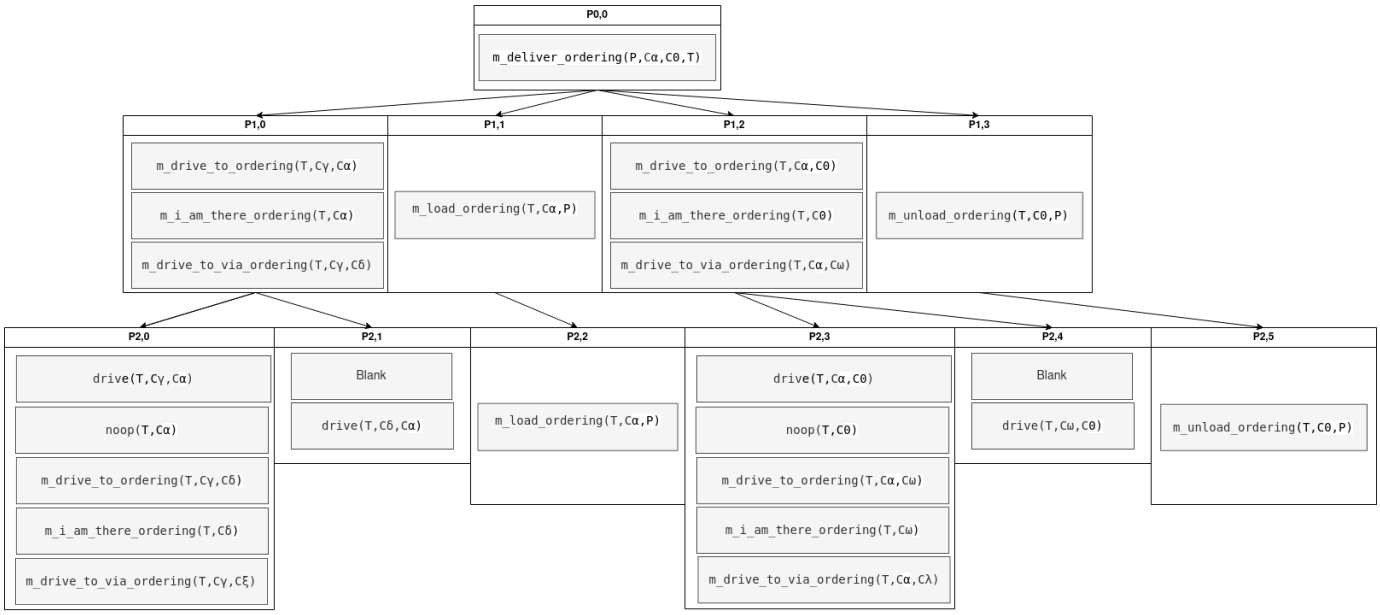

Figure 1: Example a hierarchial tree containing 3 hierarchical layers for a problem of the domain Transport as defined in the Lilotane paper. The first subtask of the method m_deliver_ordering can be accomplish by the three methods reported in the position P1,0)

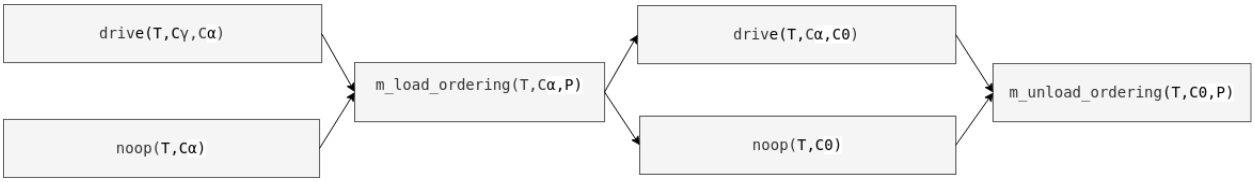

Figure 2: Space of reseach encoded by LTP into SAT clauses