The HTN Domain “Factories”

Malte Sönnichsen,[1] Dominik Schreiber[2]
Karlsruhe Institute of Technology
[1]malte@soennichsen.xyz [2]dominik.schreiber@kit.edu

Introduction

In this paper we present the benchmark domain “Factories” for Hierarchical Task Network (HTN) planning written in the HDDL format (Höller et al. 2020).

In this benchmark domain, the planning objective is to construct a *factory* by satisfying a dependency graph on certain *resources* while using *trucks* to deliver the needed resources from one place to another. As such, our domain is a crossover of traditional logistics and transport domains with a producer-consumer problem.

Overview

An input problem consists of f factories, d dependencies, and t trucks. Each factory *requires* certain resources in order to be built, which are the factory’s dependencies, and is able to *produce* certain resources when built. As such, the problem’s factories depend on each another: The construction of some factory F may require the construction of other factories which produce the resources necessary to construct F. This renders the problem recursive and makes the choice of a hierarchical planning model natural. The set of dependencies can be seen as a Directed Acyclic Graph (DAG) induced by the required and produced resources of each factory.

Transporting resources from a producer to a consumer requires trucks. There can be multiple trucks in the problem, so the tactical choice of which truck should do the delivery can lead to varying plan lengths.

In order to avoid complex conditions, we introduced composite resources which are fused from two other resources. This way, a factory has at most one resource demand in the HDDL while the number of needed actual resources can be arbitrarily high. Also, for the sake of simplicity, the resources a certain factory requires for its construction and for producing a resource itself are identical.

Generating Problems

Our problem generator script features three inputs which are (i) the number of factories, (ii) the maximum number of resource dependencies *per factory*, and (iii) the number of trucks. The graph of locations is generated as a simple undirected circle of n locations with up to n additional random edges, where n is set to twice the number of factories and trucks. Trucks are placed randomly in the graph. Dependencies are introduced to the problem by iterating over the factories “from left to right” and repeatedly adding dependencies of the current factory to a resource that some factory to the left produces. This ensures that the dependency graph is in fact acyclic and can be ordered. The first factory has no resources and the final factory is the destination object to be constructed.

```
(:method m_factory_already_constructed
    :parameters (?f - factory
                 ?l - location)
    :task (construct_factory ?f ?l)
    :precondition (factory-at ?f ?l)
    :subtasks ()
)
(:method m_construct_factory
    :parameters (?f - factory
                 ?r - resource
                 ?l - location)
    :task (construct_factory ?f ?l)
    :precondition (and
        (demands ?f ?r)
        (location-free ?l)
        (not (factory-constructed ?f))
    )
    :ordered-subtasks (and
        (get_resource ?r ?l)
        (construct ?f ?r ?l)
    )
)
```

Figure 1: The two methods which decompose the initial task `(construct_factory ?f ?l)`.

Properties

The Factories domain is recursive, i.e., it may feature task T as a possible subtask in a method for task T. The domain is totally ordered.

The domain makes use of positive and negative method preconditions. All constraints are stated conjunctively (“STRIPS-style”), i.e., as a list of literals.

We think the Factories domain is an appealing bench-

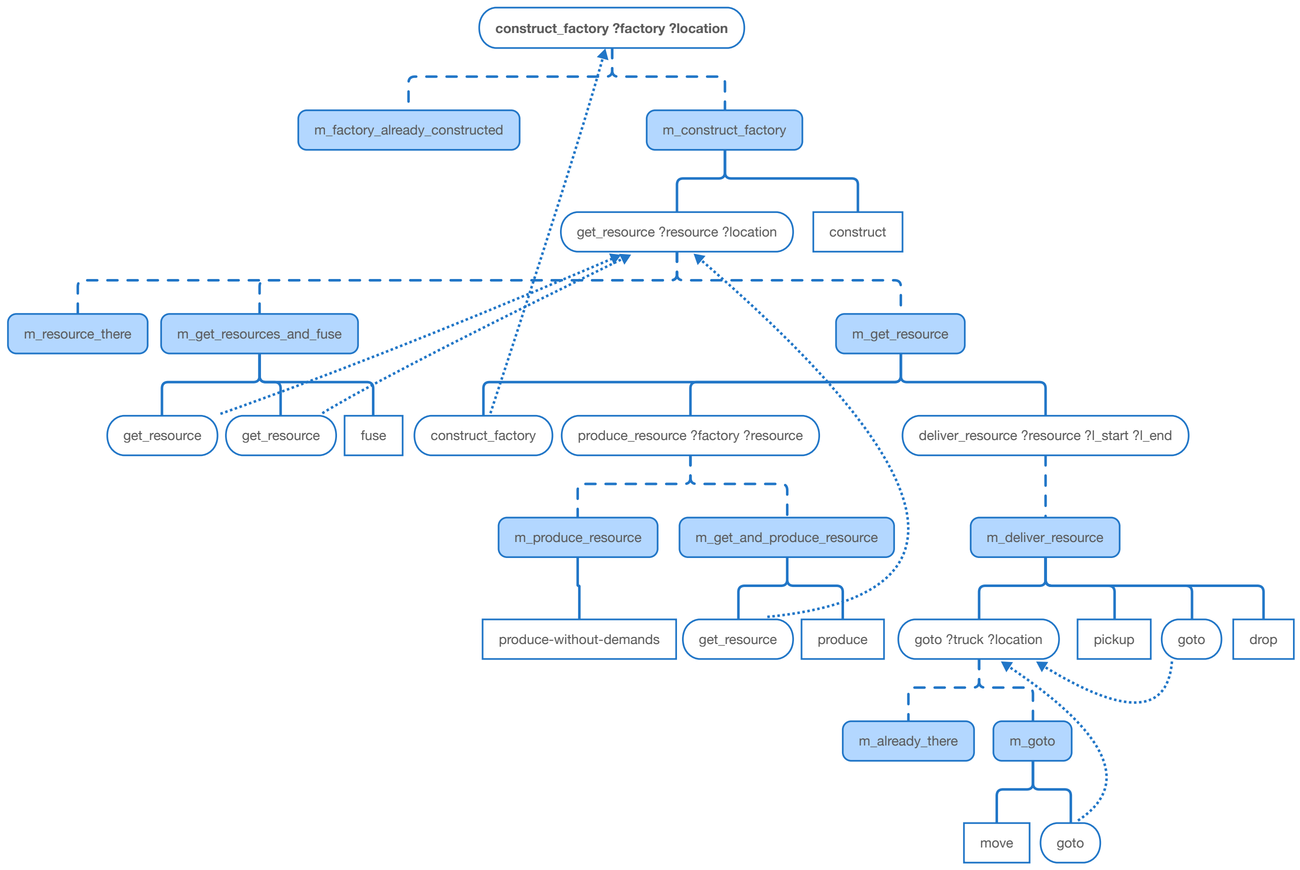

Figure 2: Illustration of the hierarchy of the Factories domain. Rectangles represent primitive tasks, circular containers represent composite tasks, and blue rectangles with rounded corners represent methods. Dashed lines denote the choice (OR) of a method, straight lines denote sequences (AND) of subtasks. Dotted arrows hint to recursive subtask relationships.

mark domain because it is a very natural application of hierarchical planning and because the problem size is easily scaleable up to dimensions which are very difficult for common HTN planners while the problem description stays relatively compact. A problem's properties can be steered using the three input parameters: Increasing the number of trucks makes grounding more difficult and produces higher numbers of methods per task. Conversely, increasing the number of factories increases the overall size of the task network, and increasing the number of dependencies makes the problem more "logically dense" while increasing the size of the task network to a moderate degree.

With our simple problem generation, the non-trivial decisions a planner has to make "only" involves the choice of which trucks should deliver which resources and which factory to place at which position; still, it is an interesting domain to benchmark planners on because it tests their ability to ground (if applicable) and search this quite straight-forward hierarchical structure as efficiently as possible. Moreover, the resulting plan length may vary considerably based on a planner's strategic decisions regarding truck assignments and factory placements, so we believe that the domain lends itself to evaluations concerning plan quality and plan optimization techniques.

Although heavily idealized, we imagine that our domain's structure can be adapted to a plethora of similar tasks that feature DAG-like dependencies.

Acknowledgments

Many thanks to Gregor Behnke for finding, and proposing fixes to, several bugs.

References

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *AAAI*, 9883–9891.