Hierarchical task network planning for multi-agent robotics

Martin Estgren
<mares480@student.liu.se>
Linköping University, Sweden

May 30, 2018

1 Introduction

There are many different approaches to automated planning varying from the classic STRIPS framework to probabilistic and temporal planning systems. In this project the performance of autonomous planning using *Heirachcial task network* (HTN) for a *multi-agent robotics domain* is examined in regards to execution time and plan quality.

The structural base for this project is the *Robot Operating System* (ROS) in conjunction with the *MORSE* simulation environment. *ROS* provides a framework for passing messages between so-called *ROS Nodes* over a networked protocol. *MORSE* is a robotics simulation environment build on top of *Blender* which provides integration with *ROS*.

Heirachcial task network planning is a large research area in of itself and has been examined both in concrete and theoretical applications, for more info on the topic, see *K. Erol et al.* [1]. The focus of this paper is on the *Simple Heirachcial Ordered Planner* (SHOP), a HTN planner developed by *D. Nau et al.* [2] as a solution to other planners computational resource requirement. SHOP has seen some evaluation in multi-agent systems [3] despite its single-agent design. A popular alternative, and predecessor to HTN planning is the STRIPS family of planners.

STRIPS is one of the oldest autonomous planning systems, first presented in 1971 by *R. Fikes and N. Nilsson*[4] and although the basic concept has been the same, there have been significant extension proposed. PDDL was presented in 1998 [5] as a attempt to unify typical planners at the time. PDDL uses world-state and operator structure similar to the one in *STRIPS*. *PDDL* has seen research within multi-agent domains, to the point oh having a language extension specifically for that purpose [6].

Similar to *PDDL*, *SHOP* have operators on the form *Precondition* followed by a list of changes which are applied to the world-state if the *Preconditions* holds for the current world-state.

In addition to *operators*, a SHOP domain is defined by a set of *tasks*. Tasks contains multiple sets of *Precondition* where each set corresponds to a sequence of *tasks* and/or *operators*. Based on these sub-sequences, the *task* is split into the two categories *composite* and *primitive*.

Tasks which only contains sequences of *operators* are categorised as *primitive* tasks. Tasks which contains at least one subtask are categorised as *composite* and serves as the structure which constrain the search procedure. The structure of *operators*, *primitive tasks*, and *composite tasks* can be seen in figure 1, 2, and 3 respectively.

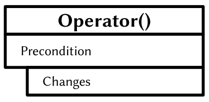

Figure 1: Structure of an *Operator*.

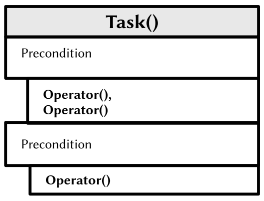

Figure 2: Structure of an *Primitive Task*.

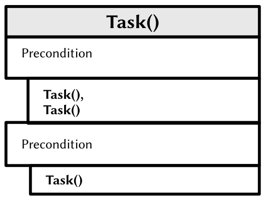

Figure 3: Structure of an *Composite Task*.

What differentiates *SHOP* from other *HTN* planners is its *search procedure*, where a *depth-first* approach is used, resulting in plan generation moving from the initial world-state towards the goal-state. This is different to the *partial-ordered* plans produced by other planners at the time [2]. An advantage with this method is that at any moment during the planning procedure, the world-state is fully known. In figure 4 an abstract representation of the way *SHOP* performs a search is presented.

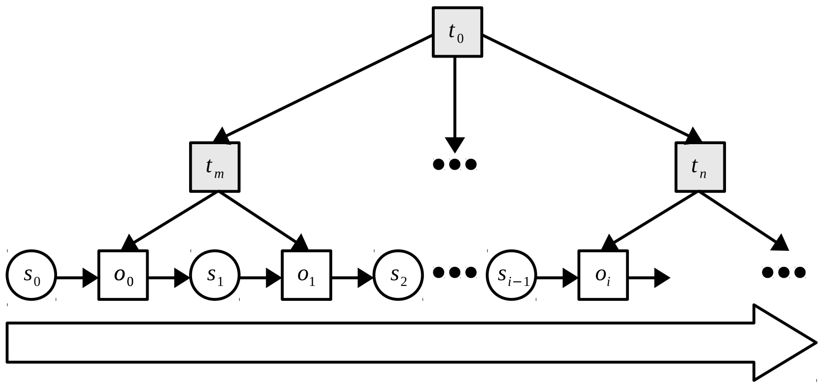

Figure 4: Example of a *SHOP* search procedure. Circles represents world states and actions (o) are produced in the order of the large arrow.

2 Method

In this section the ROS integration, planning system, and evaluation method. The ROS integration part describes custom ROS nodes and the planning system describes the planning domain which serves as base for the result and discussion sections.

ROS integration

Three separate ROS nodes are developed for the purpose of this project. Each responsible for a high level function required to simulate the *search and rescue scenario.*

- *Quad Controller* - Acts as internal controller for a single quad using a *state machine.*
- *Quad Planner* - Provides plans for available quads. Interfaces with *pyhop* for plan production.
- *Location Broadcast* - Keeps track of spatial regions which serves as *victims* in the search and rescue scenario.

The *ROS Topics* each node interacts with can be seen in figure 5, 6, and 7 respectively. Arrows going into a node represents *Topic Subscriptions*, outgoing *Topic Publishers*, and bidirectional *ROS Services.*

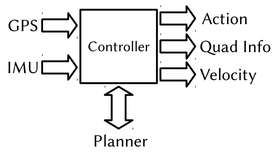

Figure 5: Topic interface for *QuadController.*

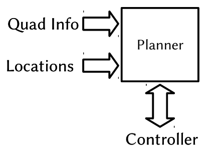

Figure 6: Topic interface for *QuadPlanner.*

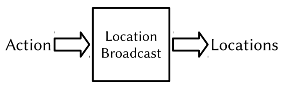

Figure 7: Topic interface for *LocationBroadcast.*

Domain definition

Support for autonomous planning is already integrated into the ROS ecosystem though the package ROSPlan [7] but it is limited to PDDL 2.1 compatible planners. For the purpose of this project, SHOP is integrated into ROS using *pyhop*, an implementation of SHOP in *Python* by *D. Nau.* One significant advantage to *pyhop* compared to the original SHOP planner is that the symbolic inference engine is traded for the full expressivity of the *Python* programming language. The implemented planning domain is based on the *Emergency Services Logistics Domain* used in the course *TDDD48* for their lab-series.

Three *operators* are defined for the planning domain:

- *Load* - Quad loads one of some resource into its cargo hold.
- *Move* - Quad moves to a given location.
- *Unload* - Quad unloads on of some resource from its cargo hold.

All plans produced by the planner are defined as a sequence of *operators*, this is done since the planner system is unable to directly affect the world state and has to act thought the quads.

In addition to the *operators*, a set of *tasks* are defined:

- *Deliver with quad* - Pick a *location* which *needs* a given resource and execute the sub-sequence *Move*, *Load for*, *Move*, *Unload for.*

- *Load for* - Load a given quad with as much as said quad can carry or until the needs of the target location is fulfilled by the cargo of the quad.
- *Unload for* - Unload cargo from a given quad until either the locations needs have been fulfilled or the quad is empty.

For a full overview of the produced planning domain, see *Appendix A*.

Performance Evaluation

Evaluation of the planning system is done by timing the time required for the planner to solve a given problem. This is done though the construction of a set of problem scenarios where the planner tasked with solving. The *SHOP* planner is compared with a planner working with a relaxed planning domain. The plans produced in the relaxed domain follow the same structure as the complete domain but the amount of resources each quad is delivering to the given locations is arbitrary.

3 Results

In this section we present the results produced during the project. The first section presents a comparison between the *SHOP* planner and baseline planner.

Performance comparison

To examine performance, 4 different scenarios were use, varying in number of quads and locations. Each scenarios is executed two times, one for the SHOP planner and one for the baseline planner. In the table 1 the raw data for each scenario can be observed.

Table 1: Raw data for performance comparison.

Time	Quads	Locations	Capacity	Planner	Scenario
19.1518	4	6	4	Pyhop	1
23.1751	4	6	4	Random	1
35.7737	2	6	4	Pyhop	2
62.7879	2	6	4	Random	2
10.7904	2	3	4	Pyhop	3
19.9757	2	3	4	Random	3
21.9313	8	8	4	Pyhop	4
25.5463	8	8	4	Random	4

In figure 8 the relative performance of the planners can be observed. The *Quads*, *Locations*, and *Capacity* columns are omitted from the figure since they are the same for both planners.

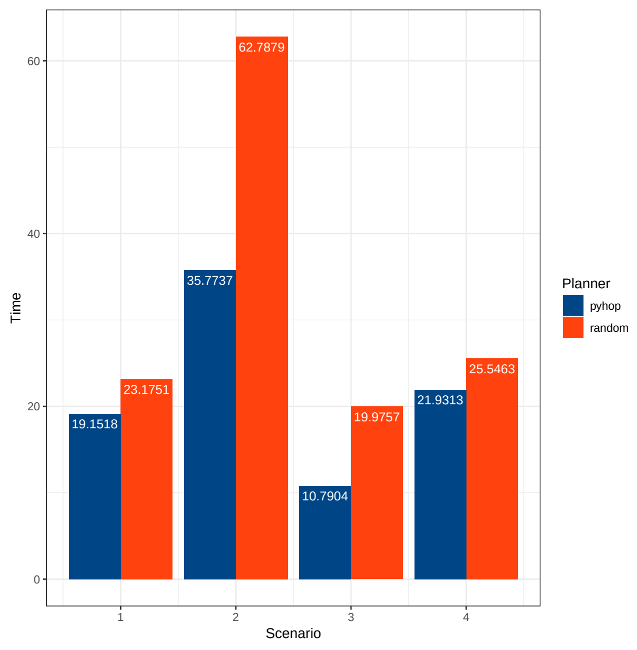

Figure 8: Graph showing the difference between the *SHOP* and baseline planner. Each group of bars represents a planning problem and each bar represents the time taken for the different planners to complete said problem.

Plan production

An example of a plan produced by the planner is available in the listing 1. The plan involves 4 quads which are tasked with delivering a variable amount of resources (water) to 4 different locations.

```
Move quad1 to depot
Load quad1 with water
Move quad2 to depot
Load quad2 with water
Move quad3 to depot
Load quad3 with water
Load quad1 with water
Move quad0 to depot
Load quad0 with water
Load quad3 with water
Load quad1 with water
Load quad0 with water
Load quad3 with water
Load quad1 with water
Load quad0 with water
Load quad3 with water
Load quad0 with water
Move quad0 to location3
Unload water from quad0 at location3
Unload water from quad0 at location3
Unload water from quad0 at location3
Unload water from quad0 at location3
Move quad1 to location4
Unload water from quad1 at location4
Unload water from quad1 at location4
Unload water from quad1 at location4
Unload water from quad1 at location4
Move quad3 to location2
Unload water from quad3 at location2
Unload water from quad3 at location2
Unload water from quad3 at location2
Unload water from quad3 at location2
Move quad2 to location4
Unload water from quad2 at location4
Move quad0 to depot
Load quad0 with water
Move quad1 to depot
Load quad1 with water
Load quad1 with water
Load quad1 with water
Move quad0 to location3
Unload water from quad0 at location3
Move quad1 to location1
Unload water from quad1 at location1
Unload water from quad1 at location1
Unload water from quad1 at location1
```

Listing 1 Example of a plan produced by *SHOP*.

4 Discussion

In this section findings related to the project are presented followed by a conclusion and potential future work.

Findings

There are two primary things to take away from the results. The *SHOP* planner performs better in comparison to the baseline (as seen in figure 8). For half of the scenarios presented, the time difference between the planners were a couple of percent but for scenarios where the number of quads were limited (scenario 2 and 3), *SHOP* performed significantly better. The second significant observation is that even if the planning system is sequential in nature, interleaved and efficient plans can be produced as a result of the quads not travelling between locations instantaneous.

Conclusion

During this project we have examined the performance of the *SHOP* planner for planning in a multi-agent robotics domain. The result shows a decrease in time required to fulfil a set of requirement for the given domain compared to a simpler planning system. For problem scenarios where there is a excess of quads compared to amount of resource which needs to be transported, both planners perform close to each other.

Going into this project we hypothesised that the planning system would aid in structuring the cooperating segment between agent and this is the result observed in the results. Additionally, the planning system allows for seemingly complex cooperating without significant changes to the on-board quad controller.

Future work

The reason for using a "relaxed" planning domain for the comparison is a result of a completely random task execution for the agent resulted in minutes long execution times, even for simple domains. This result was expected and not very informative. The best evaluation procedure would be to compare the *SHOP* planner with one in the *ROSPlan* package with regards to both plan execution but also time taken to produce the plans.

The current implementation may not produce good multi-agent plans for domains where mutually exclusive resources occurs. Such resources can result in plans that are completely sequential since each agent is waiting for its turn with the resource. The domain implemented in this project does not suffer from this limitation since resources can be produced ad infinitum at the depot. Other papers have come to similar conclusions, for example [3] where *SHOP* is integrated with the *IMPACT* environment but the planner produces plans in a sequential order. Possible workaround is to model the distributed nature of multi-agent planning in the planning domain, resulting in a more complex domain but with potentially better execution performance.

References

[1] Kutluhan Erol, James Hendler, and Dana S Nau. Complexity results for htn planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[2] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, pages 968–973. Morgan Kaufmann Publishers Inc., 1999.

[3] Jürgen Dix, Héctor Muñoz-Avila, Dana S Nau, and Lingling Zhang. Impacting shop: Putting an ai planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence*, 37(4):381–407, 2003.

[4] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[5] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. 1998.

[6] Dániel László Kovács. A multi-agent extension of pddl3. 1. 2012.

[7] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *ICAPS*, pages 333–341, 2015.

[8] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice.* Elsevier, 2004.

[9] Oliver Obst, Anita Maas, and Joschka Boedecker. Htn planning for flexible coordination of multiagent team behavior. *Fachberichte Informatik*, pages 3–2005, 2005.

[10] Ronald Alford, Ugur Kuter, and Dana S Nau. Translating htns to pddl: A small amount of domain knowledge can go a long way. In *IJCAI*, pages 1629–1634, 2009.

[11] Raphaël Lallement, Lavindra De Silva, and Rachid Alami. Hatp: An htn planner for robotics. *arXiv preprint arXiv:1405.5345*, 2014.

A Search and Rescue *SHOP* domain

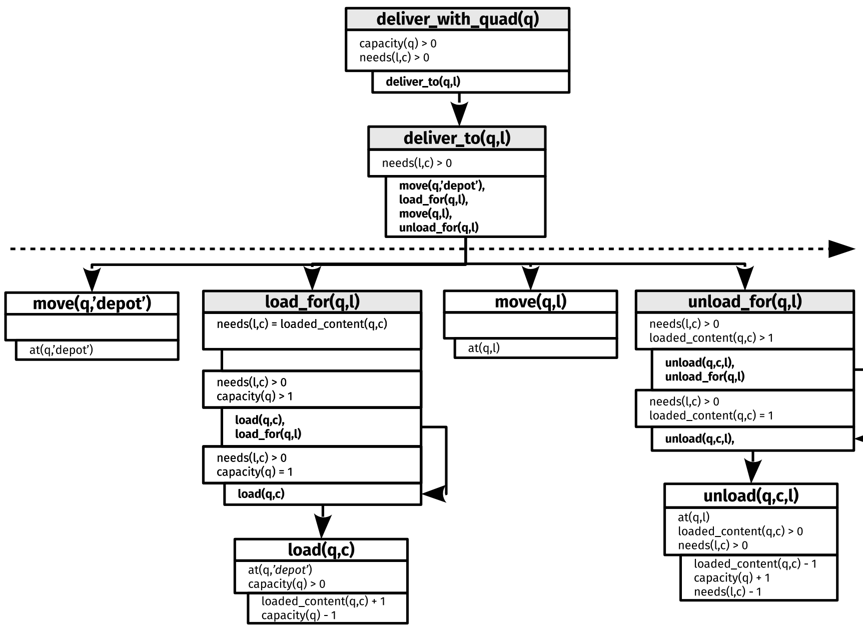

Figure 9: Search and Rescue domain implemented for *SHOP*