OptiPlan - a CSP-based partial order HTN planner

Oleksandr Firsov Humbert Fiorino Damien Pellier

Univ. Grenoble Alpes - LIG
Grenoble, France
oleksandr.firsov humber.fiorino damien.pellier @univ-grenoble-alpes.fr

Introduction

In this paper, we introduce OptiPlan, a planner that participated in partial-order HTN track of IPC 2023. OptiPlan is a hierarchical planner capable of solving partial-order problems by encoding them as CSPs [1] and generating partial order solution plans.

Encoding resolution techniques, particularly SAT encodings, have proved their worth in various IPC competitions. Solving HTN problems via CSP was little studied [2] compared to SAT [3]. These techniques, however, offer a number of advantages over SAT: (1) CSP encoding provides a natural way of expressing numerical constraints, which is not possible with SAT; (2) CSP encoding lets naturally express constraints on method decompositions (logical or numerical) and (3) CSP techniques are much more mature than SAT, and are covered my multiple industrial-level solvers[4].

The most distinct, property of OptiPlan is its capability to produce partial-ordered solution plans. The reasoning behind this feature is twofold. First, in various contexts, oftentimes industrial, it is crucial to make plans as flexible as possible, as it helps anticipate unforeseeable events [5][6]. A natural answer to this are partial-ordered solutions, which allow shifting tasks without any impact on the quality of the plan. Second, unlike deordering of total-order plans [7] which cannot guarantee quality, or optimality, of the produced plans, OptiPlan can guarantee these properties.

Optiplan Principle

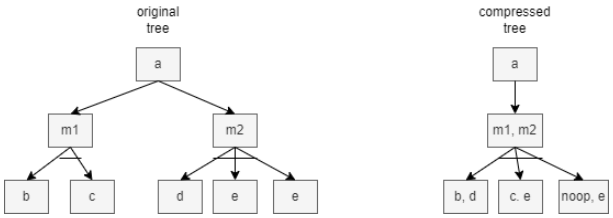
OptiPlan is based upon Task Decomposition Graph (TDG) [8] structure and hybrid planning formalization [9], which combines the concepts of HTN with POCL [10]. The difference being that our search space is a tree, instead of graph, where:

- **OR** nodes represent possible method decompositions
- **AND** nodes represent abstract and primitive tasks

Only tasks can be *leaves*, and only primitive tasks are considered *terminal* leaves.

Constructing a complete search space is infeasible in HTN planning[11], so OptiPlan operates using an iterative deepening search. Initially, our tree consists of an artificial root and its children: abstract tasks of initial HTN, as well as dummy primitive tasks t_0 and t_∞, that correspond to the initial and goal states. Along with the tree, we keep track of ordering constraints introduced by initial task network and method decompositions.

Figure 1: Example of TDG compression



In this search space, we attempt to find a subtree, such that:

1. It has exclusively terminal leaves (i.e., plan is concrete)
2. Every precondition of the subtree has a causal link supporting it (i.e., no open goals)
3. There are no threats on the causal links
4. There are no conflicting ordering constraints

If a solution can't be found, it means that the search space is not big enough to support it. So we update the search space by expanding the non-terminal leaves of the tree, and try to solve the problem again. This process is repeated until either a solution is found, or failure termination conditions are met (e.g., there are no abstract leaves left), in which case problem is deemed unsolvable.

Implementation

As our planner requires a grounded instance of the problem, we use PDDL4J[12] to parse, pre-process, and instantiate it.

To fully benefit from numeric variables, we perform a compression procedure on the tree, where we attempt to merge mutex nodes from the same depth level into a single node. This can be best explained on a simple example in Fig. 1, where we compress two methods *m1, m2* of an abstract task a.

To find the solution, OptiPlan uses Chuffed[13] as its CSP solver.

References

[1] S. C. Brailsford, C. N. Potts, and B. M. Smith, "Constraint satisfaction problems: Algorithms and applications," *European Journal of Operational Research*, vol. 119, no. 3, pp. 557–581, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221798003646

[2] P. Surynek and R. Barták, "Encoding htn planning as a dynamic csp," in *Principles and Practice of Constraint Programming-CP 2005: 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005. Proceedings 11*. Springer, 2005, pp. 868–868.

[3] I. Georgievski and M. Aiello, "Htn planning: Overview, comparison, and beyond," *Artificial Intelligence*, vol. 222, pp. 124–156, 2015.

[4] J.-F. Puget, "Constraint programming next challenge: Simplicity of use," in *Principles and Practice of Constraint Programming–CP 2004: 10th International Conference, CP 2004, Toronto, Canada, September 27-October 1, 2004. Proceedings 10*. Springer, 2004, pp. 5–8.

[5] D. Liu, H. Wang, C. Qi, P. Zhao, and J. Wang, "Hierarchical task network-based emergency task planning with incomplete information, concurrency and uncertain duration," *Knowledge-Based Systems*, vol. 112, pp. 67–79, 2016.

[6] D. Liu, H. Li, J. Wong, and M. Khallaf, "Hierarchical task network approach for time and budget constrained construction project planning," *Technological and Economic Development of Economy*, vol. 25, pp. 1–24, 04 2019.

[7] C. Muise, S. McIlraith, and C. Beck, "Optimally relaxing partial-order plans with maxsat," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 22, 2012, pp. 358–362.

[8] P. Bercher, G. Behnke, D. Höller, and S. Biundo, "An admissible htn planning heuristic." in *IJCAI*, 2017, pp. 480–488.

[9] S. Biundo and B. Schattenberg, "From abstract crisis to concrete relief—a preliminary report on combining state abstraction and htn planning," in *Sixth European Conference on Planning*, 2001.

[10] D. McAllester and D. Rosenblatt, "Systematic nonlinear planning," 1991.

[11] R. Alford, V. Shivashankar, U. Kuter, and D. Nau, "Htn problem spaces: Structure, algorithms, termination," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 3, no. 1, 2012, pp. 2–9.

[12] D. Pellier and H. Fiorino, "Pddl4j: a planning domain description library for java," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 30, no. 1, pp. 143–176, 2018.

[13] G. Chu, P. J. Stuckey, A. Schutt, T. Ehlers, G. Gange, and K. Francis. (2016) Chuffed, a lazy clause generation solver. https://github.com/chuffed/chuffed.