Odin: A Planner Based on Saturated Transition Cost Partitioning

Dominik Drexler, Jendrik Seipp, David Speck

Linköping University, Linköping, Sweden
⟨dominik.drexler, jendrik.seipp, david.speck⟩@liu.se

This planner abstract describes an optimal classical planner called *Odin*. Odin uses A* search (Hart, Nilsson, and Raphael 1968) with an admissible heuristic (Pearl 1984) based on abstraction heuristics and saturated transition cost partitioning (Keller et al. 2016) to find optimal plans. Odin's main strength is in tasks where optimal plans contain the same actions multiple times, which is often the case in transportation domains.

Introduction

Odin is an optimal classical planner and implements a new version of our work on subset-saturated transition cost partitioning (Drexler, Speck, and Mattmüller 2020; Drexler, Seipp, and Speck 2021). The planner is built on top of Fast Downward (Helmert 2006) and Scorpion (Seipp, Keller, and Helmert 2020) and runs A* (Hart, Nilsson, and Raphael 1968) with saturated transition cost partitioning over two types of abstraction heuristics: 1. Cartesian abstraction heuristics over goal and landmark subtasks (Seipp and Helmert 2018) computed with the batch refinement strategy from Speck and Seipp (2022) and 2. Pattern databases (Haslum et al. 2007) for systematic patterns.

Saturated Transition Cost Partitioning

Saturated transition cost partitioning is a method for admissibly combining the information of a collection of abstraction heuristics (Keller et al. 2016; Drexler, Seipp, and Speck 2021). Given an ordered set of abstraction heuristics, it assigns to each heuristic a fraction of the remaining costs to preserve its heuristic estimates, and leaves the remaining costs for subsequent heuristics.

The remaining costs are represented as a transition cost function $tcf : \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ that maps state-operator pairs (i.e., transitions) to real-valued costs. Since the number of transitions is exponential in the size of the input task, the performance of the method depends heavily on a compact representation of tcf. We use binary decision diagrams (BDDs) (Bryant 1986) from the CUDD library (Somenzi 2015) to compactly represent sets of states associated with the same cost value.

A special case of a transition cost function is an operator cost function $ocf : \mathcal{O} \to \mathbb{R}$, which maps operators to real-valued costs. Odin uses operator cost functions as a fallback solution when using transition cost functions is too time or memory intensive. More precisely, we use transition cost functions when the number of transitions in an abstraction heuristic is less than 40K.

The quality of a saturated cost partitioning heuristic depends heavily on the order of the heuristics. Therefore, it is beneficial to compute saturated cost partitionings over multiple orders and evaluates over the resulting estimates. Since orders are optimized for a particular state, we compute new orders in an online manner during the search (Seipp 2021).

Keeping the heuristic estimates of all states is sometimes wasteful. Therefore, we preserve the heuristic estimates of only the subset of states that are within a fixed perimeter of the goal (Seipp and Helmert 2019). The perimeter is the distance from the initial state to a goal in the abstraction. To make subtraction of transition cost functions more efficient, we also subtract no costs for transitions to unsolvable states (Drexler, Seipp, and Speck 2021).

Operator Pruning Techniques

We use two operator pruning techniques, one as preprocessing and one during the search. As preprocessing, after grounding the input task, we apply the h^2 preprocessor from Alcázar and Torralba (2015), which prunes spurious actions and simplifies the task. During the search we use strong stubborn set pruning (Alkhazraji et al. 2012; Wehrle and Helmert 2014; Röger et al. 2020). However, since the effectiveness of strong stubborn set pruning depends strongly on the domain and sometimes the computation does not pay off, we disable the pruning if less than 20% of the successor states are pruned after 1000 expansions.

Configurations

Odin uses a different configuration depending on the PDDL language features that are present after grounding and simplifying the task.

No Conditional Effects and No Axioms. Odin runs saturated transition cost partitioning over CARTESIAN and SYS-SCP abstraction heuristics.

Conditional Effects and No Axioms. Odin runs Scorpion's saturated operator cost partitioning implementation over the SYS-SCP abstraction heuristics. In principle, Odin

can be extended to work with conditional effects, but this does not work out of the box because conditional effects introduce an additional level of state dependency.

Axioms. Odin runs uniform cost search, i.e., A* with the blind heuristic.

Post-competition Evaluation

Odin tied with Scorpion (Seipp 2023) for second place in the optimal track of the 2023 International Planning Competition, just behind the portfolio planner Ragnarok (Drexler et al. 2023). In the following, we analyze the performance of Odin and in particular compare its performance to saturated *operator* cost partitioning.

Despite being awarded the second place with Scorpion, Odin achieved slightly lower overall coverage (1 task) compared to Scorpion, which uses saturated operator cost partitioning. The natural question is whether it paid off in the competition to use the more expressive transition cost functions over operator cost functions to get potentially more informative heuristics at a higher computational cost. To answer this question, we conducted a post-competition analysis. For this analysis, we use the normalized domains where available and exclude RUBIKS-CUBE since Odin does not support conditional effects. This leaves us with six domains of 20 tasks each. We compare Odin to a version of Odin that uses operator cost functions, with all other components configured identically.

Table 1 shows that both planners solve the same number of tasks (62) in our post-competition experiments, and that each solves one task that the other does not.

Table 2 compares their heuristic estimates for the initial state and reveals that the estimates are almost always the same. Saturated *transition* cost partitioning (STCP) computes higher heuristic estimates for the initial state than saturated *operator* cost partitioning (SOCP) in 5 tasks, while the opposite is the case in 2 tasks.

Part of the reason for the similar performance is that the STCP planner version often uses saturated *operator* cost partitioning. Table 3 shows the fraction of abstraction heuristics that used *operator* cost partitioning. We see that in LABYRINTH, RECH.-ROBOTS-NORM and RICOCHET-ROBOTS, STCP is (almost) never used. In SLITHERLINK-NORM, only STCP was used (1.00) and yielded better estimates for the initial state. In QUANTUM-LAYOUT, STCP was also heavily used (0.96) but this never resulted in better heuristic estimates. Inspecting the found plans, we see that none of them contains the same action multiple times. This gives saturated transition cost partitioning fewer opportunities to successfully reuse costs in different contexts.

We conclude that, empirically, none of the algorithms outperforms the other on the competition domains. Saturated transition cost partitioning has worst-case exponential performance, which makes it less robust on unseen tasks or domains compared to saturated operator cost partitioning. However, saturated transition cost partitioning has the potential to produce more informative heuristics.

Domain	SOCP	STCP
FOLDING-NORM	8	8
LABYRINTH	3	3
QUANTUM-LAYOUT	13	**14**
RECH.-ROBOTS-NORM	13	13
RICOCHET-ROBOTS	**19**	18
SLITHERLINK-NORM	6	6
Overall	62	62

Table 1: Number of solved tasks by saturated operator (SOCP) and saturated transition cost partitioning (STCP) in the domains with neither conditional effects nor axioms.

Domain	$h^{SOCP} > h^{STCP}$	$h^{SOCP} < h^{STCP}$
FOLDING-NORM	0	0
LABYRINTH	0	0
QUANTUM-LAYOUT	0	0
RECH.-ROBOTS-NORM	**1**	0
RICOCHET-ROBOTS	0	0
SLITHERLINK-NORM	2	**5**
Overall	3	**5**

Table 2: Comparison of the number of times that the heuristic estimate for the initial state of saturated operator cost partitioning is larger than that of saturated transition cost partitioning and vice versa.

Domain	$\frac{\#STCP}{\#STCP+\#SOCP}$
FOLDING-NORM	0.51
LABYRINTH	0.00
QUANTUM-LAYOUT	0.96
RECH.-ROBOTS-NORM	0.10
RICOCHET-ROBOTS	0.05
SLITHERLINK-NORM	1.00
Overall	0.14

Table 3: Fraction of abstraction heuristics that were computed with saturated *transition* cost partitioning. The remaining heuristics were computed with saturated *operator* cost partitioning.

Acknowledgments

We would like to thank all the contributors to Fast Downward and Álvaro Torralba and Vidal Alcázar for the h^2 preprocessor. Special thanks to Daniel Fišer and Florian Pommerening for organizing the competition.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. This research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In *Proc. ICAPS 2015*, 2–6.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A Stubborn Set Algorithm for Optimal Planning. In *Proc. ECAI 2012*, 891–892.

Bryant, R. E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8): 677–691.

Drexler, D.; Gnad, D.; Höft, P.; Seipp, J.; Speck, D.; and Ståhlberg, S. 2023. Ragnarok. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Drexler, D.; Seipp, J.; and Speck, D. 2021. Subset-Saturated Transition Cost Partitioning. In *Proc. ICAPS 2021*, 131–139.

Drexler, D.; Speck, D.; and Mattmüller, R. 2020. Subset-Saturated Transition Cost Partitioning for Optimal Classical Planning. In *ICAPS Workshop on Heuristics and Search for Domain-independent Planning*, 23–31.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proc. AAAI 2007*, 1007–1012.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Keller, T.; Pommerening, F.; Seipp, J.; Geißer, F.; and Mattmüller, R. 2016. State-dependent Cost Partitionings for Cartesian Abstractions in Classical Planning. In *Proc. IJCAI 2016*, 3161–3169.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Röger, G.; Helmert, M.; Seipp, J.; and Sievers, S. 2020. An Atom-Centric Perspective on Stubborn Sets. In *Proc. SoCS 2020*, 57–65.

Seipp, J. 2021. Online Saturated Cost Partitioning for Classical Planning. In *Proc. ICAPS 2021*, 317–321.

Seipp, J. 2023. Scorpion 2023. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *JAIR*, 62: 535–577.

Seipp, J.; and Helmert, M. 2019. Subset-Saturated Cost Partitioning for Optimal Classical Planning. In *Proc. ICAPS 2019*, 391–400.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *JAIR*, 67: 129–167.

Somenzi, F. 2015. CUDD: CU Decision Diagram Package – Release 3.0.0. https://github.com/ivmai/cudd. Accessed: 2023-02-20.

Speck, D.; and Seipp, J. 2022. New Refinement Strategies for Cartesian Abstractions. In *Proc. ICAPS 2022*, 348–352.

Wehrle, M.; and Helmert, M. 2014. Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies. In *Proc. ICAPS 2014*, 323–331.