Dofri: Planner Abstract

Paul Höft, David Speck, Jendrik Seipp

Linköping University, Linköping, Sweden
⟨paul.hoft,david.speck,jendrik.seipp⟩@liu.se

Abstract

Cost partitioning is the foundation of today's strongest heuristics for optimal classical planning. Many cost partitioning algorithms use linear programs (LPs) to compute the heuristic value. In practice, it is often too time-consuming to solve a separate LP for each state, so it is necessary to approximate the heuristic. Our planner, Dofri, uses a refined version of the Saturated Post-hoc Optimization heuristic that reduces the computational cost without approximating the heuristic. This is done by simplifying the LPs and avoiding duplicate computations so that we can precisely compute the heuristics for each state without sacrificing heuristic quality.

Introduction

Dofri is an optimal planner submitted to the *optimal track* of the International Planning Competition (IPC) 2023 that is based on the Scorpion planning system (Seipp, Keller, and Helmert 2020), which in turn is based on the Fast Downward planning system 22.12 (Helmert 2006). Our planner, Dofri, uses an improved version of Saturated Post-hoc Optimization (SPhO) that we call lazy SPhO. It reduces the computational costs of the SPhO heuristic, not by approximation, as done for similar heuristics (Karpas, Katz, and Markovitch 2011; Seipp, Keller, and Helmert 2020), but by simplifying and skipping some of the costly linear program (LP) computations without losing heuristic quality. The improvements of lazy SPhO can be summarized as three simple but collectively significant enhancements:

1. Ignore abstractions with useless *minimum saturated cost functions*.
2. Group abstractions with the same *minimum saturated cost functions*.
3. Avoid duplicate LP calculations.

Saturated Post-hoc Optimization

Post-hoc optimization (Pommerening, Röger, and Helmert 2013) is an operator-counting heuristic that can also be interpreted as a cost partitioning heuristic. It dominates the canonical heuristic, which is based on pattern collections (Haslum et al. 2007), and is less informed but faster to compute than optimal cost partitioning (Pommerening et al. 2015). Saturated Post-hoc Optimization is an improved variant introduced by Seipp, Keller, and Helmert (2021) that combines it with *Saturated Cost Partitioning* (Seipp, Keller, and Helmert 2020) that results in a strict improvement in theory and practice. In the following, we briefly motivate and summarize the idea behind Saturated Post-hoc Optimization, followed by the optimized computation we use in Dofri.

Saturated Post-hoc Optimization

Saturated Post-hoc Optimization (SPhO), as an operator counting heuristic, computes an LP that minimizes the amount that each action must be used to to satisfy the given abstraction heuristic.

As an abstraction-based heuristic SPhO is computed over a set of abstractions H that each constraint the use of the labels l. For operator counting these labels correspond to the operators in the given task. The heuristic optimizes the operator count Y_l for each operator with respect to the restrictions derived from the abstractions by optimizing the SPhO-LP:

Definition 1 (h$^{\mathbf{SPhO}}$ Seipp, Keller, and Helmert (2021)) *$h^{SPhO}(s)$ is the objective value of the SPhO-LP:*

$$\begin{aligned} &\textit{minimize} \sum_{l \in L} cost(l) Y_l \textit{ s.t} \\ &\sum_{l \in L} mscf_h(l) Y_l \geq h(cost, s) \textit{ for all } h \in H \qquad (1) \\ &Y_l \geq 0 \textit{ for all } l \in L \end{aligned}$$

Simplification of the linear program

SPhO uses the *minimum saturated cost function* of abstractions instead of the operator cost to calculate the heuristic values. We have found that this often leads to duplicate or even completely zero-cost functions. This is interesting because the cost functions solely define the coefficient matrix of the SPhO-LP. Equal cost functions produce duplicate constraints that differ only in their bound, the abstract goal distance. This means that the tightest constraint dominates all other constraints with the same cost function making all other duplicate constraints redundant. Lazy SPhO groups all abstractions with the same *minimum saturated cost function* into sets $\bigcup_{j=1}^{m} H_j = H$ and uses only their maximum to obtain a more compact LP that is therefore easier to solve, see Definition 2.

Definition 2 *(Höft, Speck, and Seipp 2023) The* grouped *SPhO LP is:*

$$\begin{aligned} & \textit{minimize} \sum_{\ell \in L} cost(\ell) \cdot Y_\ell \textit{ s.t.} \\ & \sum_{\ell \in L} mscf_h(\ell) \cdot Y_\ell \geq \max_{h \in H_j} h(s) \textit{ for } 1 \leq j \leq m \\ & Y_\ell \geq 0 \textit{ for all } \ell \in L \end{aligned} \tag{2}$$

In addition, Dofri removes all abstractions with zero-valued *minimum saturated cost functions* as these are useless for SPhO to further simplify the LP structure.

Avoidance of Redundant LP Computations

The second type of improvement is based on the observation that only the abstract goal distance, highlighted in red in Eq. (1), changes between calls of the SPhO-LP. The abstract goal distance for states will often be the same, and if two states have the same abstract goal distance for all $h \in H$ their SPhO-LPs are the same and it is redundant to recompute this LP. Lazy SPhO therefore stores previously computed SPhO-LPs as a mapping of abstract goal distance tuple to heuristic value: $\langle h_1(cost, s), \ldots, h_n(cost, s) \rangle \rightarrow h^{SPhO}(s)$ to avoid redundant LP computations.

Implementation Details

We use the h^2 preprocessor from Alcázar and Torralba (2015) included in Scorpion to simplify planning tasks after grounding. For the abstractions, we use the *Sys-SCP* pattern selection algorithm (Seipp 2019) and *Cartesian* abstractions (Seipp and Helmert 2018) with the batch refinement strategy from Speck and Seipp (2022). If the task is found to have conditional effects, we use only the *Sys-SCP* patterns with explicit abstractions. The actual search is an A* search (Hart, Nilsson, and Raphael 1968) using the lazy Saturated Post-hoc Optimization heuristic described. We solve the SPhO-LP using CPLEX 22.11.[1]

Post-Competition Analysis

Dofri placed 4th out of 22 planners in the optimal classical track. Table 1 shows the competition coverage results of Dofri and the number of instances in which Dofri ran out of time or memory, resulting in not solving the planning problem at hand. The results suggest that, after improving the runtime of SPhO with the described changes, memory is a more limiting factor than time. Table 2 shows that the main cause of not solving a problem was running out of memory during search. Therefore, instead of only trying to speed up heuristic computation further, future research could also look into reducing the memory consumption of lazy SPhO and heuristic search in general, e.g., by making heuristics more informative or storing open and closed states more compactly.

[1]https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

Domain	Coverage	TO	MO
Folding	8	1	11
Folding-norm	8	2	10
Labyrinth	5	4	11
Quantum-Layout	13	7	0
Recharging-Robots	13	1	6
Recharging-Robots-norm	13	4	3
Ricochet-Robots	17	3	0
Rubiks-Cube	10	0	10
Rubiks-Cube-norm	10	0	10
Slitherlink	0	0	20
Slitherlink-norm	4	11	5

Table 1: Per domain competition results of Dofri. Coverage represents the number of solved instances out of 20 per domain. "TO" and "MO" denote timeouts and out-of-memory, respectively, and indicate the number of instances where time or memory constraints were the primary reason our planner could not solve the respective problem.

	Translation		Search	
	TO	MO	TO	MO
Sum	17	31	16	55

Table 2: Total number of timeouts (TO) and out-of-memory (MO) cases instances, grouped by whether the error occurred during translation or search.

Conclusions

Dofri is a new classical optimal planner in the sense that it has never been part of a previous International Planning Competition. However, the planners it is based on, Scorpion and Fast Downward, have participated with great success. The key idea of Dofri is to perform an A* search with an optimized Saturated Post-hoc Optimization heuristic, which simplifies and avoids costly but redundant LP computations.

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.

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.

Höft, P.; Speck, D.; and Seipp, J. 2023. Sensitivity Analysis for Saturated Post-hoc Optimization in Classical Planning. In *Proc. ECAI 2023*, 1044–1051.

Karpas, E.; Katz, M.; and Markovitch, S. 2011. When Optimal Is Just Not Good Enough: Learning Fast Informative Action Cost Partitionings. In *Proc. ICAPS 2011*, 122–129.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In *Proc. AAAI 2015*, 3335–3341.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In *Proc. IJCAI 2013*, 2357–2364.

Seipp, J. 2019. Pattern Selection for Optimal Classical Planning with Saturated Cost Partitioning. In *Proc. IJCAI 2019*, 5621–5627.

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

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

Seipp, J.; Keller, T.; and Helmert, M. 2021. Saturated Post-hoc Optimization for Classical Planning. In *Proc. AAAI 2021*, 11947–11953.

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