ECAI 2024
U. Endriss et al. (Eds.)

© 2024 The Authors.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/FAIA240465

Dissecting Scorpion: Ablation Study of an Optimal Classical Planner

Jendrik Seipp

Linköping University, Sweden
jendrik.seipp@liu.se

Abstract. Currently, one of the predominant approaches for optimal classical planning is A* search with heuristics that partition action costs among several abstractions of the input planning task. One example of this approach is the Scorpion planner, which computes *saturated* cost partitionings over projections and Cartesian abstractions. Scorpion participated in the International Planning Competition 2023 and achieved the second place in the optimal track. It was only outperformed by the Ragnarok portfolio planner, which includes Scorpion as a component. In this invited paper for the ECAI *Frontiers in AI* series, we present the components of Scorpion and analyze their contributions to the overall performance in an ablation study. As a result, the paper serves as a short introduction to many of the techniques that are vital for state-of-the-art performance in optimal classical planning.

1 Introduction

Classical planning is the task of finding a sequence of actions that transforms a fully observable initial state into a goal state using deterministic actions with known effects [9]. In the *optimal* setting, the task is to find a plan that minimizes the sum of the action costs. Currently, there are two main approaches to optimal classical planning: symbolic search [41] and heuristic search [3]. Both approaches aim to combat the state-space explosion problem. While symbolic search does so by compactly representing sets of states and transitions between them, heuristic search uses a heuristic function to focus the search on promising parts of the state space. While there have been attempts to combine symbolic and heuristic search [8, 7], so far the two approaches have been mostly pursued independently. The combination is challenging because even perfect heuristic information can blow up the data structures underlying symbolic search [40].

In this invited paper for the ECAI *Frontiers in AI* series, we focus on heuristic search for optimal classical planning and present a state-of-the-art representative of this approach: the Scorpion planner. By dissecting Scorpion, we provide a brief introduction to many of the techniques that are necessary for state-of-the-art performance in optimal classical planning.

Throughout the paper, we use the name *Scorpion* to refer to the planner configuration (*Scorpion 2023*) we submitted to the sequential optimization track of the International Planning Competition (IPC) 2023. However, Scorpion is also the name of the planning system that we used to implement this configuration. The Scorpion planning system is an extension of Fast Downward [13] and includes many additional search algorithms and heuristics.[1] The Scorpion planner, which participated in the IPC 2023, uses A* [10] with an admissible heuristic [21] to find optimal plans. The overall heuristic is based on *abstraction heuristics* that are combined by *saturated cost partitioning* [36]. Furthermore, it uses *pruning techniques* to reduce the search space and improve the runtime performance. We discuss these components in the following sections.

2 Abstraction Heuristics

Arguably the most important component of Scorpion are the abstraction heuristics. Several types of abstractions have been proposed for classical planning, including projections [4, 6], domain abstractions [15, 19], Cartesian abstractions [2, 31], and merge-and-shrink abstractions [5, 39], listed in increasing order of generality. They all have in common that they reduce the size of the state space by aggregating states and thus make it possible to precompute abstract goal distances, which can then be used as heuristic estimates in the A* search.

Scorpion uses a combination of projections, a.k.a. pattern database (PDB) heuristics (SYS-SCP), and Cartesian abstractions (CART).[2] In both cases, a single abstraction is usually not sufficient to solve the task optimally. Therefore, we build several abstractions and combine them using *cost partitioning* [17, 43]. By dividing action costs among the abstractions, cost partitioning ensures that the sum of the admissible heuristic values remains admissible. This still leaves the question of how to generate the abstractions and which cost partitioning method to use.

The simplest method for generating PDB heuristics is to consider all subsets of variables (each subset is called a *pattern*) and compute a pattern database for each of them, possibly limiting the number of variables per pattern to obtain reasonably-sized abstractions. This naive method can be improved by considering only *interesting* patterns [22], i.e., those sets of variables that cannot be decomposed into two smaller sets without losing any information. In addition to this systematic enumeration, there is also a local search algorithm that iteratively generates larger patterns (iPDB) [12]. Recently, we introduced another method for generating PDB heuristics, which is

[1] https://github.com/jendrikseipp/scorpion
[2] Since Cartesian abstractions do not support actions with conditional effects, Scorpion only uses the SYS-SCP PDBs for such tasks. If a task contains axioms after the translation phase, which converts the input PDDL task [20] to finite-domain representation [14], we do not use any abstractions and instead use the blind heuristic. For the presentation and experiments in this paper, we only consider tasks without conditional effects and without axioms.

based on Dantzig-Wolfe decomposition [24]. Here, patterns are also enumerated systematically from small to large, but only those patterns are kept that increase the optimal cost partitioning value.

In the Scorpion planner, to generate a set of useful pattern database (PDB) heuristics, we iteratively generate larger *interesting* patterns [22] and keep the ones that are *useful* [27]. To judge the utility of a PDB, we compute a cost partitioning among the PDBs already selected and store the remaining action costs. For a new pattern, we compute the induced projection and all its goal distances under this remaining cost function. A PDB computed this way is useful if it contains non-zero goal distances.

To obtain a diverse set of Cartesian abstraction heuristics, we consider Cartesian abstractions of landmark and goal task decompositions [31]. Each such abstraction simplifies the original task by focusing on achieving only a single landmark or goal. To maintain shortest paths in the abstractions with minimal effort, we use incremental search [37].

3 Cost Partitioning

The easiest way to combine abstraction heuristics is to use the maximum of the individual heuristic values. However, this approach can only be as informed as the most informed heuristic in each state. A preferable way that can yield heuristics that are more informed than any of the individual heuristics is to use *cost partitioning* [17, 43]. By dividing the costs of actions among the abstractions, cost partitioning ensures that no action cost is counted more than once in the sum of the heuristic values. As a result, the sum of the heuristic values remains admissible.

The first heuristic based on cost partitioning was the *canonical heuristic* for PDBs [12]. It is based on the notion of *independence*: two abstractions are *independent* if each action is relevant for at most one of them. The canonical heuristic computes the maximal sets of pairwise independent abstractions, sums their estimates and maximizes over the resulting heuristics. While the canonical heuristic is the most accurate heuristic computable given only the information about which actions are relevant in which abstractions, *post-hoc optimization* [22] dominates it in terms of accuracy, because it also has access to the heuristic values induced by the individual abstractions.

Two more basic cost partitioning algorithms are *uniform cost partitioning* [17] and *zero-one cost partitioning* [11]. Uniform cost partitioning divides the cost of each action evenly among the abstractions that use it, while zero-one cost partitioning assigns the full cost of an action to the first abstraction that uses it and zero cost to all others, based on some arbitrary order of the abstractions.

All of these cost partitioning algorithms are dominated by *optimal cost partitioning* [23], which can be computed in linear time for abstraction [17, 18] and landmark [16] heuristics, by solving a linear program. However, even for a moderate number of abstractions, computing an optimal cost partitioning can be computationally prohibitive. This remains the case even when using an algorithm based on Dantzig-Wolfe decomposition [24].

Three of the cost partitioning algorithms above can be improved by using the following insight: we can often reduce the costs assigned to a heuristic without lowering any heuristic values. For abstraction and landmark heuristics, we can even compute a unique minimum cost function that preserves all estimates. We call such a cost function *saturated*. By saturating the cost functions assigned to a heuristic by zero-one cost partitioning, uniform cost partitioning, or post-hoc optimization, we obtain the variants *saturated cost partitioning* [30, 36], *opportunistic uniform cost partitioning* [34], and *saturated post-hoc optimization* [38], respectively. The saturated variants dominate their non-saturated counterparts in terms of heuristic quality. While there is no dominance relation between saturated cost partitioning, saturated post-hoc optimization, and opportunistic uniform cost partitioning, saturated cost partitioning is often preferable in practice, which is why it forms an integral part of Scorpion.

4 Saturated Cost Partitioning

Given an ordered sequence of heuristics, saturated cost partitioning (SCP) iteratively assigns each heuristic h only the costs that h needs for justifying its estimates and saves the remaining costs for subsequent heuristics [36]. The quality of the resulting saturated cost partitioning heuristic strongly depends on the order in which the component heuristics are considered [33]. Additionally, we can obtain much stronger heuristics by maximizing over multiple saturated cost partitioning heuristics computed for different orders instead of using a single saturated cost partitioning heuristic [33]. We therefore compute a diverse set of SCP heuristics online during the search [28]. To this end, we select every ten-thousandth evaluated state s, compute an SCP heuristic h^{SCP} tailored to s and add it to our initially empty set of SCP heuristics if h^{SCP} yields a higher estimate for s than all previously added SCP heuristics. We limit the time for computing and adding new SCP heuristics in this way to 100 seconds.

Originally, this diversification procedure was done *offline* before the search, using states sampled with random walks from the initial state [33]. Performing the diversification *online* during the search has the advantage that the search can start right away instead of having to wait for the offline computation to finish.

4.1 Orders for Saturated Cost Partitioning

To tailor an SCP heuristic for a given state s, we order the abstractions with a greedy algorithm using the $q_{\frac{h}{stolen}}$ scoring function [26, 36]. It considers those abstractions first that have a high ratio of heuristic value for s divided by the costs that are "stolen" from other abstractions to justify this value.

4.2 Subset Saturation

Instead of preserving all estimates, we can also compute a subset-saturated cost partitioning, which only preserves the estimates for a subset of the states [32], for example only for a single state. The variant of subset-saturated cost partitioning that offers the best trade-off between the quality of the resulting heuristic and the runtime of the computation is called *perim*. It preserves all heuristic values within a given perimeter around the goal state. When computing a perim SCP heuristic for state s, we preserve all heuristic estimates smaller than or equal to the estimate for s. In Scorpion, we use the *perim** variant, which first computes a perim SCP and then uses the remaining costs to compute an SCP preserving the estimates of *all* states (under the reduced cost function).

5 Pruning Techniques

Scorpion uses two pruning techniques as another measure to combat combinatorial explosion. It computes h^2 mutexes [1] to remove irrelevant operators and atoms. This method has proven to be very effective in practice, since it often prunes a large fraction of actions and has a low runtime overhead.

Table 1. Comparison of base Scorpion planner to variants with individual components turned off. The rows show the total number of solved tasks and the number of domains where a variant solves more and less tasks than the base variant. SYS-SCP only uses the systematic SCP PDBs and drops the Cartesian abstractions. CART only uses the Cartesian abstractions and drops the SCP PDBs. MAX maximizes over abstraction heuristics instead of computing saturated cost partitionings. OFFLINE computes SCP heuristics offline instead of online. RANDOM uses random orders for SCP heuristics instead of greedy orders. ALL computes SCP heuristics that preserve the estimates of all states instead of using the perim* variant. NO-h^2 does not prune irrelevant operators. NOPOR does not compute stubborn sets for partial order reduction.

	BASE	SYS-SCP	CART	MAX	OFFLINE	RANDOM	ALL	NO-h^2	NOPOR
Coverage	1353	1300	1265	995	1354	1325	1350	1320	1345
#Domains better than BASE	0	5	3	1	8	6	6	7	3
#Domains worse than BASE	0	15	27	37	6	8	9	11	6

The second pruning technique are *stubborn sets* [42, 25], a form of *partial order reduction*. For a given state, a stubborn set is a subset of the applicable actions that guarantees that at least one of the actions starts an optimal plan. By ignoring all other applicable actions, we can reduce the branching factor without sacrificing optimality. Since stubborn sets have a non-negligible runtime overhead, we only compute them for a state if the fraction of pruned successor states is at least 20% of the total successor states after 1000 expansions.

6 Ablation Study

We now analyze the contributions of the major components of Scorpion to the overall performance by turning off each component in turn. We use the Downward Lab toolkit [35] for running this experiment and limit runtime and memory to 30 minutes and 8 GiB. Our benchmark set consists of all 1947 tasks without conditional effects from 54 domains used by the optimal sequential tracks of the IPCs 1998–2023. All benchmarks, code and experiment data are available online [29].

Tables 1 and 2 show the results of the ablation study. Since the total number of solved tasks can depend heavily on the coverage in a single domain, we focus mostly on the two bottom rows in Table 1, which show for how many domains a variant solves more or fewer tasks than the base variant. Regarding the choice of abstractions, we can see that the SYS-SCP PDBs are more important than the Cartesian abstractions, but both of them are necessary for maximum coverage. Disabling cost partitioning and instead maximizing over the abstraction heuristics (MAX) leads to the largest drop in performance among the tested variants, confirming that cost partitioning is one of the main recent innovations in classical planning. Computing SCP heuristics offline instead of online (OFFLINE) actually helps more often than it hurts performance on a per-domain basis. However, computing cost partitionings online has a much lower runtime than the offline computation with its fixed time limit [28]. Using random orders for SCP heuristics (RANDOM) instead of greedy orders only leads to a small drop in performance and it even helps in some domains. This suggests that it could be beneficial to investigate better ways for ordering heuristics for SCP. Saturating for all states instead of using the perim* variant (ALL) incurs the lowest drop in overall coverage among the tested variants. Since there are even six domains that benefit from using the simpler variant that saturates for all states, these data suggest that the perim* variant is not always necessary. Disabling the h^2 mutex computation (NO-h^2) decreases coverage for 11 domains, but increases it for 7 domains, contradicting earlier findings that h^2 mutexes are almost always beneficial [1]. Disabling stubborn sets (NOPOR) leads to a minor drop in coverage, which is probably the case since not many domains are amenable to partial order reduction in the first place. Also, partial order reduction is most useful for heuristics that are slow to compute, which is not the case for SCP heuristics.

7 Conclusions

We have presented the Scorpion planner and dissected its components in an ablation study. The results of the ablation study show that the choice of abstractions and the use of saturated cost partitioning are the most important components of Scorpion. The results also suggest that it could be worthwhile to better understand when a certain technique is beneficial and possibly enable or disable it on a per-task basis.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and 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

[1] V. Alcázar and Á. Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Proc. ICAPS 2015*, pages 2–6, 2015.

[2] T. Ball, A. Podelski, and S. K. Rajamani. Boolean and Cartesian abstraction for model checking C programs. In *Proc. TACAS 2001*, pages 268–283, 2001.

[3] B. Bonet and H. Geffner. Planning as heuristic search. *AIJ*, 129(1):5–33, 2001.

[4] J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[5] K. Dräger, B. Finkbeiner, and A. Podelski. Directed model checking with distance-preserving abstractions. *Software Tools for Technology Transfer*, 11(1):27–37, 2009.

[6] S. Edelkamp. Planning with pattern databases. In *Proc. ECP 2001*, pages 84–90, 2001.

[7] D. Fišer, Á. Torralba, and J. Hoffmann. Operator-potential heuristics for symbolic search. In *Proc. AAAI 2022*, pages 9750–9757, 2022.

[8] S. Franco, Á. Torralba, L. H. S. Lelis, and M. Barley. On creating complementary pattern databases. In *Proc. IJCAI 2017*, pages 4302–4309, 2017.

[9] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[11] P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for domain-independent planning. In *Proc. AAAI 2005*, pages 1163–1168, 2005.

[12] P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI 2007*, pages 1007–1012, 2007.

[13] M. Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

[14] M. Helmert. Concise finite-domain representations for PDDL planning tasks. *AIJ*, 173:503–535, 2009.

Table 2. Number of solved tasks by the base Scorpion planner and the variants with individual components turned off. For a description of the variants, see Table 1.

Coverage	Base	Sys-SCP	Cart	Max	Offline	Random	All	No-h^2	NoPOR
agricola (20)	**4**	**4**	3	3	**4**	**4**	**4**	3	**4**
airport (50)	45	**46**	42	27	42	40	44	37	44
barman (34)	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
blocksworld (35)	28	28	28	21	29	**30**	28	28	28
childsnack (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
data-network (20)	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
depots (22)	**14**	**14**	12	7	**14**	**14**	**14**	13	**14**
driverlog (20)	**15**	**15**	14	12	**15**	**15**	**15**	**15**	**15**
elevators (50)	**44**	43	43	39	**44**	**44**	**44**	**44**	**44**
floortile (40)	32	**34**	16	21	**34**	32	**34**	15	32
folding (20)	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
freecell (80)	70	32	**72**	21	71	59	70	71	70
ged (20)	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**
grid (5)	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
gripper (20)	**8**	7	**8**	**8**	**8**	**8**	**8**	**8**	**8**
hiking (20)	**19**	**19**	14	16	**19**	**19**	16	**19**	**19**
labyrinth (20)	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
logistics (63)	38	35	**39**	19	38	38	37	**39**	37
miconic (150)	145	145	**146**	55	145	144	145	145	145
movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
mprime (35)	31	**34**	30	29	33	32	32	32	31
mystery (30)	**19**	**19**	**19**	18	**19**	**19**	**19**	**19**	**19**
nomystery (20)	**20**	**20**	**20**	15	**20**	**20**	**20**	**20**	**20**
openstacks (100)	57	**58**	57	51	**58**	55	**58**	**58**	**58**
organic (20)	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
organic-split (20)	**14**	**14**	**14**	**14**	**14**	**14**	**14**	10	**14**
parcprinter (50)	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	44
parking (40)	17	16	14	1	17	**18**	17	16	17
pathways (30)	**5**	**5**	**5**	4	**5**	**5**	**5**	**5**	**5**
pegsol (50)	**50**	**50**	48	48	**50**	**50**	**50**	**50**	**50**
petri-net (20)	**8**	**8**	7	2	**8**	**8**	**8**	**8**	**8**
pipes-nt (50)	26	24	26	20	**27**	**27**	26	**27**	26
pipes-t (50)	**19**	18	18	16	18	18	18	**19**	**19**
psr (50)	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
quantum-layout (20)	**14**	**14**	13	7	**14**	**14**	**14**	**14**	**14**
recharging-robots (20)	**14**	**14**	13	12	**14**	**14**	**14**	13	**14**
ricochet-robots (20)	17	**19**	13	17	17	17	18	18	18
rovers (40)	**13**	**13**	11	9	**13**	**13**	**13**	**13**	12
satellite (36)	11	10	9	7	11	**12**	11	**12**	10
scanalyzer (50)	**35**	**35**	23	21	33	**35**	33	**35**	**35**
slitherlink (20)	6	6	4	6	**7**	6	6	5	6
snake (20)	14	14	13	**15**	**15**	14	**15**	14	14
sokoban (50)	**50**	**50**	**50**	48	**50**	**50**	**50**	**50**	**50**
spider (20)	**18**	15	15	11	**18**	**18**	16	17	**18**
storage (30)	**16**	**16**	**16**	15	**16**	**16**	**16**	**16**	**16**
termes (20)	**14**	13	12	**14**	**14**	13	**14**	**14**	**14**
tetris (17)	**13**	**13**	11	11	**13**	**13**	**13**	11	**13**
tidybot (40)	**33**	32	**33**	28	**33**	**33**	**33**	30	**33**
tpp (30)	**12**	**12**	8	6	**12**	**12**	**12**	**12**	**12**
transport (70)	38	36	34	36	37	38	**42**	38	38
trucks (30)	18	13	16	10	17	**19**	17	18	18
visitall (40)	31	30	18	14	30	18	30	31	**32**
woodworking (50)	**50**	**50**	**50**	38	**50**	**50**	**50**	**50**	49
zenotravel (20)	**14**	13	**14**	9	**14**	13	13	**14**	**14**
Sum (1947)	1353	1300	1265	995	**1354**	1325	1350	1320	1345

[15] I. T. Hernádvölgyi and R. C. Holte. Experiments with automatically created memory-based heuristics. In *Proc. SARA 2000*, pages 281–290, 2000.

[16] E. Karpas and C. Domshlak. Cost-optimal planning with landmarks. In *Proc. IJCAI 2009*, pages 1728–1733, 2009.

[17] M. Katz and C. Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In *Proc. ICAPS 2008*, pages 174–181, 2008.

[18] M. Katz and C. Domshlak. Optimal admissible composition of abstraction heuristics. *AIJ*, 174(12–13):767–798, 2010.

[19] R. Kreft, C. Büchner, S. Sievers, and M. Helmert. Computing domain abstractions for optimal classical planning with counterexample-guided abstraction refinement. In *Proc. ICAPS 2023*, pages 221–226, 2023.

[20] D. McDermott. The 1998 AI Planning Systems competition. *AI Magazine*, 21(2):35–55, 2000.

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

[22] F. Pommerening, G. Röger, and M. Helmert. Getting the most out of pattern databases for classical planning. In *Proc. IJCAI 2013*, pages 2357–2364, 2013.

[23] F. Pommerening, M. Helmert, G. Röger, and J. Seipp. From non-negative to general operator cost partitioning. In *Proc. AAAI 2015*, pages 3335–3341, 2015.

[24] F. Pommerening, T. Keller, V. Halasi, J. Seipp, S. Sievers, and M. Helmert. Dantzig-Wolfe decomposition for cost partitioning. In *Proc. ICAPS 2021*, pages 271–280, 2021.

[25] G. Röger, M. Helmert, J. Seipp, and S. Sievers. An atom-centric perspective on stubborn sets. In *Proc. SoCS 2020*, pages 57–65, 2020.

[26] J. Seipp. Better orders for saturated cost partitioning in optimal classical planning. In *Proc. SoCS 2017*, pages 149–153, 2017.

[27] J. Seipp. Pattern selection for optimal classical planning with saturated cost partitioning. In *Proc. IJCAI 2019*, pages 5621–5627, 2019.

[28] J. Seipp. Online saturated cost partitioning for classical planning. In *Proc. ICAPS 2021*, pages 317–321, 2021.

[29] J. Seipp. Code and data for the ECAI 2024 paper "Dissecting Scorpion: Ablation Study of an Optimal Classical Planner". https://doi.org/10.5281/zenodo.13361692, 2024.

[30] J. Seipp and M. Helmert. Diverse and additive Cartesian abstraction heuristics. In *Proc. ICAPS 2014*, pages 289–297, 2014.

[31] J. Seipp and M. Helmert. Counterexample-guided Cartesian abstraction refinement for classical planning. *JAIR*, 62:535–577, 2018.

[32] J. Seipp and M. Helmert. Subset-saturated cost partitioning for optimal classical planning. In *Proc. ICAPS 2019*, pages 391–400, 2019.

[33] J. Seipp, T. Keller, and M. Helmert. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In *Proc. AAAI 2017*, pages 3651–3657, 2017.

[34] J. Seipp, T. Keller, and M. Helmert. A comparison of cost partitioning algorithms for optimal classical planning. In *Proc. ICAPS 2017*, pages 259–268, 2017.

[35] J. Seipp, F. Pommerening, S. Sievers, and M. Helmert. Downward Lab. https://doi.org/10.5281/zenodo.790461, 2017.

[36] J. Seipp, T. Keller, and M. Helmert. Saturated cost partitioning for optimal classical planning. *JAIR*, 67:129–167, 2020.

[37] J. Seipp, S. von Allmen, and M. Helmert. Incremental search for counterexample-guided Cartesian abstraction refinement. In *Proc. ICAPS 2020*, pages 244–248, 2020.

[38] J. Seipp, T. Keller, and M. Helmert. Saturated post-hoc optimization for classical planning. In *Proc. AAAI 2021*, pages 11947–11953, 2021.

[39] S. Sievers and M. Helmert. Merge-and-shrink: A compositional theory of transformations of factored transition systems. *JAIR*, 71:781–883, 2021.

[40] D. Speck, F. Geißer, and R. Mattmüller. When perfect is not good enough: On the search behaviour of symbolic heuristic search. In *Proc. ICAPS 2020*, pages 263–271, 2020.

[41] Á. Torralba. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. PhD thesis, Universidad Carlos III de Madrid, 2015.

[42] M. Wehrle, M. Helmert, Y. Alkhazraji, and R. Mattmüller. The relative pruning power of strong stubborn sets and expansion core. In *Proc. ICAPS 2013*, pages 251–259, 2013.

[43] F. Yang, J. Culberson, R. Holte, U. Zahavi, and A. Felner. A general theory of additive state space abstractions. *JAIR*, 32:631–662, 2008.