

Better Orders for Saturated Cost Partitioning in Optimal Classical Planning

Jendrik Seipp

University of Basel
Basel, Switzerland
jendrik.seipp@unibas.ch

Abstract

Cost partitioning is a general method for adding multiple heuristic values admissibly. In the setting of optimal classical planning, saturated cost partitioning has recently been shown to be the cost partitioning algorithm of choice for pattern database heuristics found by hill climbing, systematic pattern database heuristics and Cartesian abstraction heuristics. To evaluate the synergy of the three heuristic types, we compute the saturated cost partitioning over the combined sets of heuristics and observe that the resulting heuristic is outperformed by the heuristic that simply maximizes over the three saturated cost partitioning heuristics computed separately for each heuristic type. Our new algorithm for choosing the orders in which saturated cost partitioning considers the heuristics allows us to compute heuristics outperforming not only the maximizing heuristic but even state-of-the-art planners.

Introduction

A*-search (Hart, Nilsson, and Raphael 1968) with an admissible heuristic (Pearl 1984) is one of the most efficient methods for solving state-space search problems optimally. Since a single heuristic is often insufficient for challenging problems, it is usually desirable to combine the estimates of multiple heuristics admissibly. One way of doing so is to use their maximum, but this merely *selects* the best heuristic for each evaluated state.

Cost partitioning (Katz and Domshlak 2008; Yang et al. 2008) is a general method for actually *combining* multiple heuristics. By dividing the original operator costs among the heuristics, it allows to *sum* the heuristic estimates admissibly. The resulting cost-partitioned heuristic is often much stronger than the maximum over the heuristics.

In the setting of optimal classical planning (Ghallab, Nau, and Traverso 2004), it has been shown that an optimal cost partitioning can be computed in polynomial time for abstraction (Katz and Domshlak 2008; 2010) and landmark (Karpas and Domshlak 2009) heuristics. In practice, however, computing an optimal cost partitioning is often prohibitively expensive, even for a single state (Pommerening, Röger, and Helmert 2013). There are many algorithms for computing non-optimal cost partitionings (e.g., Haslum, Bonet, and Geffner 2005; Haslum et al. 2007; Katz and Domshlak 2008;

Pommerening, Röger, and Helmert 2013). A recent addition is saturated cost partitioning (Seipp and Helmert 2014), which operates on an ordered sequence of heuristics and iteratively assigns the minimum costs that one heuristic needs for its estimates, before using the remaining costs for subsequent heuristics.

Seipp, Keller, and Helmert (2017a) compared multiple cost partitioning algorithms theoretically and experimentally. They computed cost partitionings for pattern databases (PDBs) found by hill climbing (Haslum et al. 2007), systematic PDBs (Pommerening, Röger, and Helmert 2013), Cartesian abstractions (Seipp and Helmert 2013) and landmark heuristics (Karpas and Domshlak 2009), and showed that saturated cost partitioning is usually the method of choice for all tested heuristic types on the IPC benchmark collection. Their results also show that the different heuristic types have their strengths in different benchmark domains. This suggests that computing saturated cost partitioning heuristics for the combined sets of heuristics could yield an even more accurate heuristic. Testing this hypothesis is our first contribution.

It turns out that combining the different heuristic types in a single cost-partitioned heuristic indeed yields a very strong heuristic. We observe that a simple maximization over three separately computed saturated cost partitioning heuristics solves even more tasks, which motivates our second contribution: by introducing a new algorithm for choosing the orders in which saturated cost partitioning considers the heuristics, we are able to compute heuristics that not only close the gap to the simple maximization heuristic, but also outperform the previous state of the art in optimal classical planning.

Background

Since cost partitioning can be applied to any collection of heuristics for state-space search, our definitions are not specific to classical planning. A *state space* is a directed, labeled graph $\mathcal{T} = \langle S, \mathcal{L}, c, T, s_I, S_\star \rangle$, where S is a finite set of *states*; \mathcal{L} is a finite set of *labels*; $c : \mathcal{L} \mapsto \mathbb{R}$ is a (possibly negative) *cost function*; T is a set of labeled and weighted *transitions* $s \xrightarrow{l, c(l)} s'$ for $s, s' \in S$ and $l \in \mathcal{L}$; $s_I \in S$ is the *initial state*; and $S_\star \subseteq S$ is the set of *goal states*. A state space is *regular* if $c(l) \geq 0$ for all labels l . We mainly con-

sider regular state spaces, but permit negative costs within cost partitionings as proposed by Pommerening et al. (2015).

The *goal distance* $h^*(s) \in \mathbb{R} \cup \{-\infty, \infty\}$ of a state $s \in S$ is the cost of a cheapest path from s to a goal state in S_* . It is ∞ if no such path exists and $-\infty$ if paths of arbitrarily low negative cost exist, which cannot happen if \mathcal{T} is regular.

We write $h^c(s) \in \mathbb{R} \cup \{-\infty, \infty\}$ for the heuristic estimate of state s under cost function c' . A heuristic is admissible if it never overestimates the true goal distance of a given state.

Cost Partitioning

A single heuristic is usually unable to capture enough information about challenging state-space search problems. Therefore, it is often beneficial to consider multiple heuristics (Holte et al. 2006). Cost partitioning makes the sum of their individual estimates admissible.

Definition 1. Cost partitioning.

Let $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ be a tuple of admissible heuristics for a regular state space $\mathcal{T} = \langle S, \mathcal{L}, c, T, s_1, S_* \rangle$. A cost partitioning over \mathcal{H} is a tuple $\mathcal{C} = \langle c_1, \dots, c_n \rangle$ of (general) cost functions whose sum is bounded by c : $\sum_{i=1}^n c_i(l) \leq c(l)$ for all $l \in \mathcal{L}$. The cost-partitioned heuristic $h^{\mathcal{C}}$ is defined as $h^{\mathcal{C}}(s) := \sum_{i=1}^n h_i^{c_i}(s)$.

The sum of the cost-partitioned heuristics remains admissible due to the way \mathcal{C} distributes the cost of each (operator) label among the heuristics.

Saturated Cost Partitioning

Even though an optimal cost partitioning for a given state can be computed in polynomial time for abstraction (Katz and Domshlak 2008; 2010) and landmark (Karpas and Domshlak 2009) heuristics, the computation usually takes prohibitively long (Pommerening, Röger, and Helmert 2013). Seipp and Helmert (2014) introduced saturated cost partitioning, an algorithm for quickly computing suboptimal but very informative cost partitionings. It works on an ordered sequence of heuristics and iteratively assigns each heuristic only the costs that this heuristic can actually exploit, before using the remaining costs for subsequent heuristics.

Definition 2. Saturated cost partitioning.

Let \mathcal{T} be a regular state space and \mathcal{H} be a set of admissible heuristics. Given an order $\omega = \langle h_1, \dots, h_n \rangle$, the saturated cost partitioning $\mathcal{C} = \langle c_1, \dots, c_n \rangle$ and the remaining cost functions $\langle \bar{c}_0, \dots, \bar{c}_n \rangle$ are defined by

$$\begin{aligned} \bar{c}_0 &= c \\ c_i &= \text{saturate}(h_i, \bar{c}_{i-1}) \\ \bar{c}_i &= \bar{c}_{i-1} - c_i \end{aligned}$$

The saturated cost function $\text{saturate}(h, c)$ is the minimal cost function $c' \leq c$ with $h^{c'}(s) = h^c(s)$ for all states s (Seipp and Helmert 2014). For abstraction heuristics h , the saturated cost of operator o is the maximum over $h(s) - h(s')$ for all abstract state transitions $s \rightarrow s'$ induced

$h_{\text{HC}}^{\text{SCP}}$	$h_{\text{Sys}}^{\text{SCP}}$	$h_{\text{Cart}}^{\text{SCP}}$	$h_{\text{HC, Sys, Cart}}^{\text{SCP}}$	$\max(h_{\text{HC}}^{\text{SCP}}, h_{\text{Sys}}^{\text{SCP}}, h_{\text{Cart}}^{\text{SCP}})$
805	852	965	1006	1032

Table 1: Number of solved tasks for different diverse saturated cost partitioning heuristics.

by o . Seipp, Keller, and Helmert (2017b) showed that the order in which saturated cost partitioning considers the heuristics greatly influences the quality of the resulting heuristic. To find good orders they introduced a hill climbing procedure that optimizes a random initial order for a set of samples by iteratively switching the positions of two heuristics until no better order can be reached. However, they obtained the strongest heuristics by maximizing over multiple saturated cost partitioning heuristics computed for different (random) orders. They presented a diversification method that only adds an order if it yields a heuristic with a higher estimate than all previously added orders for any of a set of samples to ensure that only heuristics for “useful” orders are stored and evaluated during search.

In a follow-up paper (2017a) the same authors computed diverse saturated cost partitioning heuristics over pattern database heuristics found by hill climbing (HC), systematic pattern database heuristics (Sys) and Cartesian abstraction heuristics (Cart). They observed that none of the three resulting heuristics dominates the other two, but that each heuristic has its strengths in different benchmark domains. This suggests combining the underlying heuristics.

Combining Heterogeneous Heuristics

We evaluate this idea by running an experiment on 1667 benchmark tasks from all optimal tracks of the International Planning Competition (IPC). Like Seipp, Keller, and Helmert (2017a), we use PDB heuristics found by hill climbing in at most 60 seconds (note that they used a limit of 15 minutes), systematic PDBs of sizes 1 and 2, and Cartesian abstractions of the *landmark* and *goal* task decompositions by Seipp and Helmert (2014). These heuristics will form the basis of all experiments in this paper. The first four columns in Table 1 compare the diverse saturated cost partitioning heuristics computed over the three different heuristic types separately and the heuristic obtained by computing diverse saturated cost partitionings over the combination of the heuristics.

Combining the different heuristics in $h_{\text{HC, Sys, Cart}}^{\text{SCP}}$, solving 1006 tasks, increases the total number of solved tasks by 201, 154 and 41 tasks, compared to $h_{\text{HC}}^{\text{SCP}}$, $h_{\text{Sys}}^{\text{SCP}}$ and $h_{\text{Cart}}^{\text{SCP}}$, respectively. Each of these increases is a significant improvement in the optimal planning setting where problem difficulty tends to scale exponentially. However, the heuristic that simply maximizes over the first three separately computed heuristics, $\max(h_{\text{HC}}^{\text{SCP}}, h_{\text{Sys}}^{\text{SCP}}, h_{\text{Cart}}^{\text{SCP}})$, solves even more tasks (1032).

Appending a heuristic to the tuple of heuristics considered by saturated cost partitioning can only make the resulting estimates more accurate. Therefore, for each heuristic order in

$\max(h_{\text{HC}}^{\text{SCP}}, h_{\text{Sys}}^{\text{SCP}}, h_{\text{Cart}}^{\text{SCP}})$ using only a single type of heuristic we could easily construct a combined order that “dominates” the former. E.g., for an order ω_{HC} using only pattern database heuristics found by hill climbing, we can construct an order $\omega_{\text{HC}} \oplus \omega_{\text{Sys}} \oplus \omega_{\text{Cart}}$ by starting with the original order and appending the other heuristics in any order.

We assume that the overhead incurred by evaluating the underlying heuristics is roughly the same for $h_{\text{HC, Sys, Cart}}^{\text{SCP}}$ and $\max(h_{\text{HC}}^{\text{SCP}}, h_{\text{Sys}}^{\text{SCP}}, h_{\text{Cart}}^{\text{SCP}})$. Therefore, the difference in coverage between the two heuristics must stem from $h_{\text{HC, Sys, Cart}}^{\text{SCP}}$ using worse or fewer orders, suggesting that the procedure for finding orders can be improved.

Finding an Order for a Single State

We hypothesize that the main weakness of the diversification procedure by Seipp, Keller, and Helmert (2017b) is that it only considers random orders. As long as the number of heuristics is low enough, as it may have been the case in their experiments, a strong heuristic can often be found by diversifying random orders. Once the number of heuristics and therefore the space of orders grows too large however, we need better ways for generating orders and cannot hope to “stumble” over good ones by chance.

We believe that their procedure for optimizing a single order with a hill climbing search was a step in the right direction, but we argue that it can be improved in at least two aspects. First, the procedure tries to find a good heuristic order for a set of sample states, while the authors already proved that it can be impossible to find a single good order even for only two states. Second, their hill climbing procedure lacked an important ingredient for local optimization: a decent initial solution. To address both of these shortcomings, we propose an algorithm that greedily constructs an *initial* order for a *single* given state.

Greedy Order

Given a set of heuristics \mathcal{H} and a state s , our goal is to find an order resulting in a heuristic with an accurate estimate for s . Since all cost-partitioned heuristics are admissible and therefore underestimate the true goal distance, the ones with higher estimates are more accurate. Roughly speaking, we should therefore try to move heuristics with high estimates for s to the front of an order. However, we also have to keep in mind that only the first heuristic is allowed to use all the costs it can exploit. Subsequent heuristics operate on the costs that have not been consumed by previous heuristics. To preserve costs for as many heuristics as possible, we should therefore also “prefer” orders that begin with heuristics using few costs.

To measure how well a heuristic balances the objectives of having a high heuristic value and using few costs, we define the *value-per-cost ratio* r for a heuristic h , a cost function c and a state s as

$$r(h, c, s) = \frac{h^c(s)}{1 + \sum_{o \in \mathcal{O}} \max(0, \hat{c}_h(o))},$$

where \hat{c}_h is the saturated cost function $\text{saturate}(h, c)$ and \mathcal{O} is the set of operators of the task at hand. The denominator

Algorithm 1 Given a set of heuristics \mathcal{H} , an operator cost function c and a state s , compute a dynamic greedy order by iteratively appending the heuristic with the highest value-per-cost ratio.

```

1: function DYNAMICGREEDYORDER( $\mathcal{H}, c, s$ )
2:    $\omega \leftarrow \langle \rangle$ 
3:   while  $H \neq \emptyset$  do
4:      $h \leftarrow \arg \max_{h' \in \mathcal{H}} r(h', c, s)$ 
5:     append  $h$  to  $\omega$ 
6:      $\mathcal{H} \leftarrow \mathcal{H} \setminus \{h\}$ 
7:      $c \leftarrow c - \text{saturate}(h, c)$ 
8:   return  $\omega$ 

```

sums the positive costs that h uses. We add 1 to guarantee that the division is always defined. We ignore negative costs since preliminary experiments showed that accounting for them yields worse orders.

Algorithm 1 uses the value-per-cost ratio to greedily compute a heuristic order. Given a set of heuristics \mathcal{H} , a cost function c and a state s , it starts with an empty order ω and then iteratively appends the heuristic with the highest value-per-cost ratio and updates the remaining cost function c until all heuristics are part of ω . If there are multiple heuristics with the same value-per-cost ratio, we break ties arbitrarily.

After the cost function c has been updated, all saturated cost functions have to be recomputed for all remaining heuristics \mathcal{H} . This can become prohibitively time consuming for tasks with many or large abstractions. In our practical implementation we therefore introduce a shortcut: at the end of each loop, after updating the cost function c , we remove all heuristics h with $h^c(s) = 0$ from \mathcal{H} and collect them in the set \mathcal{H}_0 (which is empty at the beginning of the algorithm). Once the while-loop terminates, we append all heuristics from \mathcal{H}_0 to ω in an arbitrary order. This change has a negligible influence on the generated orders, but often speeds up the algorithm significantly.

Even with this modification, computing dynamic greedy orders can take several minutes for some tasks. By removing line 7 from Algorithm 1 we obtain a *static* version that precomputes the saturated cost function for each heuristic once and always returns a greedy order in less than 0.05 seconds.

Evaluation

We implemented the dynamic and static versions of our greedy algorithm in the Fast Downward planning system (Helmert 2006) and used the downward-lab toolkit (Seipp et al. 2017) to conduct experiments. As above, we evaluate our ideas on 1667 benchmark tasks from all previous optimal IPC tracks. We apply time and memory limits of 30 minutes and 2 GiB to all algorithm runs.

Single Order We begin by evaluating which algorithm produces the best initial order for a given state. For each task we compute a dynamic and static greedy order for the initial state and a random order. Table 2 compares the resulting heuristic estimates for the initial state. Ignoring optimized orders (*-opt) for now, we see that the orders found for the

	rand	static	dyn	optimized		
				rand	static	dyn
rand	–	223	105	0	55	20
static	842	–	92	184	0	17
dyn	989	501	–	268	111	0
rand-opt	1086	650	463	–	179	111
static-opt	1058	681	511	344	–	115
dyn-opt	1108	747	548	402	231	–

Table 2: Pairwise comparison of algorithms producing a single order for saturated cost partitioning. The entry in row x and column y holds the number of tasks in which algorithm x yielded a heuristic with a higher heuristic estimate for s_1 than algorithm y . For each comparison we highlight the algorithm with more such tasks in bold.

initial state by the dynamic and static greedy algorithms are much better than random orders. The results also show that the dynamic version has an edge over the static one: the former produces a better order than the latter in 501 tasks, while the opposite is the case in only 92 tasks.

Single Optimized Order Even though the greedy algorithms produce good orders on their own, Table 2 reveals that we can usually improve them. The three “optimized” versions take the non-optimized order as input and optimize them with Seipp, Keller, and Helmert’s hill climbing search (2017b) for at most three minutes. Depending on whether we use a random, static or dynamic order, 1086, 681 and 548 tasks benefit from the optimization. The results also show that optimizing the orders cancels out much of the advantage that dynamic orders have over static ones: optimized dynamic orders are better than optimized static orders in 231 tasks, but the opposite is true in 115 tasks as well. Optimizing random orders often produces higher estimates compared to using non-optimized dynamic orders (463 vs. 268 tasks). However, optimized dynamic orders are usually better than optimized random orders (402 vs. 111 tasks), showing that both an initial greedy order and optimizing it afterwards are essential for obtaining the best orders.

Diverse Orders As Seipp, Keller, and Helmert (2017b) demonstrated, we should use multiple orders and maximize over the resulting heuristics if we want to obtain accurate heuristic estimates for more than a single state. Consequently, we use their diversification procedure, but instead of random orders, we diversify optimized greedy orders. In detail, our adapted diversification procedure samples 1000 states with the sampling algorithm by Haslum et al. (2007) and then iteratively samples an additional state s , computes a greedy order ω for s , optimizes ω for s with hill climbing and adds ω to the set of orders if the corresponding heuristic has a higher heuristic estimate for any of the 1000 samples than all previously added orders. Preliminary experiments

	rand	max-rand	optimized				tasks
			rand	dyn	static	hybrid	
rand	–	5	3	5	2	2	1006
max-rand	8	–	6	4	2	2	1032
rand-opt	5	6	–	4	2	1	1009
dyn-opt	10	9	8	–	3	1	1034
static-opt	9	10	6	7	–	2	1041
hybrid-opt	10	10	8	6	3	–	1048

Table 3: Left: Pairwise coverage comparison of different heuristics derived by saturated cost partitioning. The entry in row x and column y holds the number of domains in which algorithm x solved more tasks than algorithm y . For each comparison we highlight the algorithm with higher coverage for more domains in bold. Right: Total number of solved tasks by each algorithm.

made us use a time limit of one second for optimization.

Table 3 compares four different planners that use our new diversification procedure, grouped under “optimized” and only differing in the choice of initial order, to two planners diversifying random orders. The “rand” planner corresponds to the $h_{\text{HC, Sys, Cart}}^{\text{SCP}}$ planner in the first experiment and computes diverse saturated cost partitionings over the combined set of heuristics. The “max-rand” planner corresponds to the $\max(h_{\text{HC}}^{\text{SCP}}, h_{\text{Sys}}^{\text{SCP}}, h_{\text{Cart}}^{\text{SCP}})$ planner above and maximizes over separately computed diverse saturated cost partitionings.

The results show that diversifying optimized random orders (“rand-opt”) is insufficient for solving as many tasks in total as “max-rand”, but the two algorithms are on par in a domain-wise comparison. Among the algorithms diversifying optimized orders, “static-opt” has an edge over “dyn-opt”, which in turn has an edge over “rand-opt”, both in a domain-wise comparison and in terms of total coverage. Since there are domains in which the slower-to-compute, but often more informative dynamic orders outperform static orders, we also evaluate the following hybrid algorithm: compute the dynamic order for the initial state and compute the static order for all other samples. As we can see, “hybrid-opt” combines the advantages of the two ingredients and outperforms both.

Comparison to State of the Art In our final experiment we compare the new $h_{\text{hybrid-opt}}^{\text{SCP}}$ algorithm to the symbolic search planner SymBA_2^* (Torralla, Linares López, and Borrajo 2016), the winner of the IPC 2014 sequential optimization track. SymBA_2^* uses h^2 mutexes to prune irrelevant operators (Alcázar and Torralba 2015), an important preprocessing step that can be combined with any planner. In 22 domains $h_{\text{hybrid-opt}}^{\text{SCP}}$ using h^2 mutexes solves more tasks than SymBA_2^* , while the opposite is true in only 11 domains. $h_{\text{hybrid-opt}}^{\text{SCP}}$ using h^2 mutexes also solves significantly more tasks in total (1084) than SymBA_2^* (1008).

Conclusion

We combined multiple types of abstraction heuristics in a single heuristic with saturated cost partitioning and observed a significant increase in the number of solved tasks. Analyzing the new heuristic revealed some weak spots of the heuristic construction process. As a result, we proposed a greedy algorithm for finding heuristic orders given a sample state. Combining our greedy algorithm, the hill climbing search for optimizing an order and the procedure for finding multiple diverse orders, we obtained a heuristic that outperforms the previous state of the art.

Acknowledgments

We thank Álvaro Torralba for providing us with the latest SymbA₂* version.

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project “Reasoning about Plans and Heuristics for Planning and Combinatorial Search” (RAPHPACS).

References

- Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.
- Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.
- 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 *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.
- Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 1163–1168. AAAI Press.
- Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.
- Holte, R.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170(16–17):1123–1136.
- Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733. AAAI Press.
- Katz, M., and Domshlak, C. 2008. Optimal additive composition of abstraction-based admissible heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 174–181. AAAI Press.
- Katz, M., and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence* 174(12–13):767–798.
- Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.
- Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.
- Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364. AAAI Press.
- Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.
- Seipp, J., and Helmert, M. 2014. Diverse and additive Cartesian abstraction heuristics. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. AAAI Press.
- Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. downward-lab 2.0. <https://doi.org/10.5281/zenodo.399255>.
- Seipp, J.; Keller, T.; and Helmert, M. 2017a. A comparison of cost partitioning algorithms for optimal classical planning. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*. AAAI Press.
- Seipp, J.; Keller, T.; and Helmert, M. 2017b. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3651–3657. AAAI Press.
- Torralba, Á.; Linares López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3272–3278. AAAI Press.
- Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research* 32:631–662.