Online Saturated Cost Partitioning for Classical Planning

Jendrik Seipp
Linköping University, Sweden
University of Basel, Switzerland
jendrik.seipp@liu.se

Abstract

Cost partitioning is a general method for admissibly summing up heuristic estimates for optimal state-space search. Most cost partitioning algorithms can optimize the resulting cost-partitioned heuristic for a specific state. Since computing a new cost-partitioned heuristic for each evaluated state is usually too expensive in practice, the strongest planners based on cost partitioning over abstraction heuristics precompute a set of cost-partitioned heuristics before the search and maximize over their estimates during the search. This makes state evaluations very fast, but since there is no better termination criterion than a time limit, it requires a long precomputation phase, even for the simplest planning tasks. A prototypical example for this is the Scorpion planner which computes saturated cost partitionings over abstraction heuristics offline before the search. Using Scorpion as a case study, we show that by incrementally extending the set of cost-partitioned heuristics online during the search, we drastically speed up the planning process and even often solve more tasks.

Introduction and Background

One of the main approaches for solving classical planning tasks optimally is using the A* algorithm (Hart, Nilsson, and Raphael 1968) with an admissible heuristic (Pearl 1984). Since a single heuristic usually fails to capture enough details of the planning task, it is often beneficial to compute multiple heuristics and to combine their estimates (Holte et al. 2006). The preferable method for admissibly combining heuristic estimates is cost partitioning (Haslum, Bonet, and Geffner 2005; Haslum et al. 2007; Katz and Domshlak 2008, 2010; Pommerening, Röger, and Helmert 2013). By distributing the original costs among the heuristics, cost partitioning makes the sum of heuristic estimates admissible.

Saturated cost partitioning (SCP) is one of the strongest methods for computing cost partitionings (Seipp, Keller, and Helmert 2020) and is the main component of the Scorpion planner (Seipp 2018b). SCP is based on the insight that we can often evaluate a heuristic under a reduced (action) cost function without changing any estimates. This notion is captured by so-called *saturated* cost functions. A cost function *scf* is saturated for a heuristic h, an original cost function *cost* and a subset S' of states in the planning task, if (1) *scf*$(a) \leq$ *cost*(a) for each action a and (2) for all states $s \in S'$ the heuristic estimate by h for s is the same regardless of whether we evaluate h under *cost* or *scf*. For *abstraction* heuristics (Helmert, Haslum, and Hoffmann 2007), such as the ones we consider in the experiments, we can efficiently compute a unique minimum saturated cost function.

Algorithm 1 Compute a saturated cost partitioning over an ordered sequence of heuristics ω for a cost function *cost*.

```
1: function SaturatedCostPartitioning(ω, cost)
2:     C ← ⟨⟩
3:     for each h ∈ ω do
4:         scf ← saturate_h(cost)
5:         append scf to C
6:         cost(a) ← cost(a) − scf(a) for all actions a
7:     return C
```

Algorithm 1 shows how the SCP algorithm computes saturated cost functions that form a cost partitioning of a given cost function *cost* over an ordered sequence of heuristics ω. The algorithm starts by computing a saturated cost function for the first heuristic h in ω (line 4). Afterwards, it iteratively subtracts the costs given to h from the original costs (line 6) and considers the next heuristic until all heuristics have been treated this way. The sequence of computed saturated cost functions forms the resulting cost partitioning C. We write h^{SCP}_{ω} for the cost partitioning heuristic that results from applying the SCP algorithm to the heuristic order ω.

A saturated cost partitioning can be tailored to a state s by (1) ordering the heuristics in a way that is favorable for s (Seipp, Keller, and Helmert 2020), and (2) including s in the set of states S' for which we preserve the heuristic estimates, but otherwise keeping S' small (Seipp and Helmert 2019). Since both both of these adjustments are state-of-the-art for SCP heuristics we use them for all our experiments.[1]

Offline Diversification of SCP Heuristics Most of the previous work on the topic precomputes SCPs *offline*, i.e., before the search and then computes the maximum over the SCP heuristic estimates for a given state during the search.

[1] In detail, we order the heuristics greedily with the $\frac{h}{stolen}$ scoring function (Seipp, Keller, and Helmert 2020) and use the *perim** saturator (Seipp and Helmert 2019).

Algorithm 2 Offline diversification. Find a diverse set of heuristic orders Ω for SCP before the search.

```
1: function OFFLINEDIVERSIFICATION
2:     Ω ← ∅
3:     Ŝ ← sample 1000 states
4:     repeat
5:         s ← sample state
6:         ω ← greedy order for s
7:         if ∃s' ∈ Ŝ : h^SCP_ω(s') > sup_{ω'∈Ω} h^SCP_ω'(s') then
8:             Ω ← Ω ∪ {ω}
9:     until time spent in function ≥ T
10:    return Ω
```

Algorithm 2 shows the strongest offline SCP algorithm from the literature (Seipp, Keller, and Helmert 2020). It samples 1000 states $\hat{S}$ with random walks (line 3) and then iteratively samples a new state s (line 5), computes a greedy order ω for s (line 6) and keeps ω if it is *diverse*, i.e., h^{SCP}_ω yields a higher heuristic estimate for any of the samples in $\hat{S}$ than all previously stored orders (lines 7–8). (The supremum of the empty set is $-\infty$.) The offline *diversification* procedure stops and returns the found set of orders Ω after reaching a given time limit. This last characteristic is the main drawback of the algorithm: the A* search can only start after the offline diversification finishes and so far there is no good stopping criterion except for a fixed time limit. Seipp, Keller, and Helmert (2020) showed that a limit of 1000 seconds leads to solving the highest number of IPC benchmarks in 30 minutes, but such a huge time limit obviously bloats the solving time for many tasks, especially for those that blind search would solve instantly.

Online Computation of SCP Heuristics Instead of precomputing SCP heuristics before the search, we can also compute them *online*, i.e., during the search. This approach, which we call *online-all*, computes a greedy order and the corresponding SCP heuristic for each state evaluated during the search. By design, online-all can start the A* search immediately and it has access to the states that are actually evaluated by A* and not only to randomly sampled states like the offline diversification procedure. As a result, the online-all method has been shown to be the strongest cost partitioning method for landmark heuristics (Seipp, Keller, and Helmert 2017). However, computing an SCP over abstraction heuristics for each evaluated state slows down the heuristic evaluation so much that the online variant solves much fewer tasks than precomputed SCP heuristics (Seipp, Keller, and Helmert 2020). This kind of result is typical for optimal classical planning: more work per evaluated state often results in better estimates but does not outweigh the slower evaluation speed (e.g., Karpas, Katz, and Markovitch 2011; Seipp, Pommerening, and Helmert 2015).

Online Diversification of SCP Heuristics

In this work, we combine ingredients of the offline and online-all variants to obtain the benefits of both, i.e., fast solving times and high total coverage. More precisely, we interleave heuristic diversification and the A* search: for a subset of the evaluated states, we compute a greedy order and store the corresponding SCP heuristic if it yields a more accurate estimate for the state at hand than all previously stored SCP heuristics.

Algorithm 3 Online diversification. Simultaneously diversify a set of orders Ω for SCP and compute the maximum over all induced SCP heuristic values for a given state s.

```
1: function COMPUTEHEURISTIC(Ω, s)
2:     if SELECT(s) and time spent in function < T then
3:         ω ← greedy order for s
4:         if h^SCP_ω(s) > sup_{ω∈Ω} h^SCP_ω(s) then
5:             Ω ← Ω ∪ {ω}
6:     return max_{ω∈Ω} h^SCP_ω(s)
```

Algorithm 3 shows pseudo-code for the approach, which adapts the COMPUTEHEURISTIC function used to evaluate a state. Before COMPUTEHEURISTIC is called for the first time, we initialize the set of heuristic orders Ω for SCP to be the empty set.[2] When evaluating a state s, we let the state selection function SELECT decide whether to use s for diversifying Ω (line 2). We discuss several state selection functions below, but all of them select the initial state for diversification. If s is selected, we compute a greedy order ω for s (line 3) and check whether ω induces an SCP heuristic h^{SCP}_ω with a higher estimate for s than all previously stored orders (line 4). If that is the case, we store ω (line 5). Finally, we return the maximum heuristic value for s over all SCP heuristics induced by the stored orders (line 6).

In contrast to offline diversification, this online diversification algorithm allows the A* search to start immediately. Also, online diversification can judge the utility of storing an order based on states that are actually evaluated during the search instead of basing this decision on randomly sampled states. Compared to the online-all method, online diversification evaluates states much faster.

Time Limit

Even if we compute SCPs for only a subset of evaluated states, these computations can be very costly. Therefore, we use a time limit T to ensure that the diversification stops eventually and only select a state for diversification (line 2) if the total time spent in COMPUTEHEURISTIC is less than T. The time limit also allows us to perform two optimizations: after precomputing all SCP heuristics, we can delete all abstract transition systems, since during the search we only need the abstraction functions, which map from concrete to abstract states. Furthermore, for abstractions that never contribute any heuristic information under the set of stored orders, we can even delete the corresponding abstraction functions (Seipp 2018a). While both optimizations often greatly

[2]Note that we could initialize Ω with a set of orders diversified offline. However, exploratory experiments showed that this only has a mild advantage over pure offline and pure online variants, so we only consider the pure variants here.

reduce the memory footprint, the latter also speeds up the heuristic evaluation since we need to map the concrete state to its abstract counterpart for fewer abstractions.

State Selection Strategies

We now discuss three instantiations of the SELECT function, i.e., strategies for choosing the states for which to diversify the set of orders.

Interval The first strategy selects every i-th evaluated state for a given value of i. The motivation for this strategy is to distribute the time for diversification across the state space, in order to select states for diversification that are different enough from each other to let the corresponding SCP heuristics generalize to many unseen states. Note that for i=1 this strategy selects all states until hitting the diversification time limit T. For i=1 and T=∞ the resulting heuristic dominates the online SCP variant without diversification (online-all), because both heuristics compute the same SCP heuristic for the currently evaluated state, but the variant with diversification also considers all previously stored orders.

Novelty This strategy makes the notion of "different states" explicit by building on the concept of *novelty* (Lipovetzky and Geffner 2012). Novelty is defined for factored states spaces, i.e., where each state s is defined by a set of atoms (atomic propositions) that hold in s. The novelty of a state s is the size of the smallest conjunction of atoms that is true in s and false in all states previously evaluated by the search. For a given value of k, the novelty strategy selects a state if it has a novelty of at most k.

Bellman The last strategy selects a state s if the maximum over the currently stored SCP heuristics h^{SCP}_Ω violates the Bellman optimality equation (1957) for s and its successor states, i.e., if $h^{\text{SCP}}_\Omega(\mathit{cost}, s) < \min_{s \xrightarrow{a} s' \in T}(h^{\text{SCP}}_\Omega(\mathit{cost}, s') + \mathit{cost}(a))$, where T is the set of transitions in the planning task. Whenever the Bellman optimality equation is violated for a state s, we know that the current estimate for s is lower than the true goal distance of s, in which case it seems prudent to select s for diversification.

Experiments

We implemented online diversification for saturated cost partitioning in the Fast Downward planning system (Helmert 2006) and used the Downward Lab toolkit (Seipp et al. 2017) for running experiments on Intel Xeon Silver 4114 processors. Our benchmark set consists of all 1827 tasks without conditional effects from the optimal sequential tracks of the International Planning Competitions 1998–2018. We limit time by 30 minutes and memory by 3.5 GiB. All benchmarks, code and experiment data have been published online (Seipp 2021).

For the heuristic set on which SCP operates, we use the combination of pattern databases found by hill climbing (Haslum et al. 2007), systematic pattern databases of sizes 1 and 2 (Pommerening, Röger, and Helmert 2013) and Cartesian abstractions of *landmark* and *goal* task decompositions (Seipp and Helmert 2018). When evaluating a planning algorithm, we focus on its *coverage* (number of solved tasks) and its *time score* (used for the agile track of IPC 2018). The time score of a planner P for a task that P solves in t seconds is defined as $1 - \frac{\log(t)}{\log(1800)}$. It is 0 if P fails to solve P within 1800 seconds. The total coverage and time score of a planner is the sum of its scores over all tasks.

When we diversify the set of orders online, the heuristic estimate of a state can increase between its generation and expansion. Since it is slightly preferable to reevaluate states before expanding them (Seipp 2020), we use this setting in all experiments below.

	interval						novelty		bm
	1	10	100	1K	10K	100K	1	2	
T=1000s	1152	1157	1157	1157	**1159**	1156	1153	1158	1146
T=∞	813	964	1064	1112	1140	**1149**	1140	1064	1001
T=1000s	596.3	733.5	835.6	892.7	**911.5**	901.5	853.0	721.8	779.6
T=∞	571.5	720.5	829.8	890.6	**910.5**	901.1	851.9	714.2	770.1

Table 1: Coverage (top) and time score (bottom) of different state selection strategies with and without time limit.

Evaluation of State Selection Strategies

In the first experiment, we compare the different instantiations of the SELECT function. Table 1 holds results for the interval strategy with different intervals, the novelty strategy for k=1 and k=2 and the Bellman strategy (bm). With a diversification time limit of 1000 seconds, we see that overall coverage is similar for all interval and novelty variants (1152–1159 solved tasks) and that the Bellman strategy solves fewer tasks in total than the other strategies. We obtain the highest total coverage and time score by selecting every ten thousandth evaluated state (*interval-10K*) and therefore we use this strategy in all other experiments.

Evaluation of Time Limit

Table 1 also confirms that we need a time limit for the online diversification. For all state selection strategies total coverage decreases when the time limit of 1000 seconds for the diversification is lifted (T=∞). The coverage loss is higher, the more states we may select for diversification. For example, the coverage of the *novelty-1* variant only decreases by 13 tasks, because the number of selected states is limited by the number of atoms A in the planning task. For *novelty-2* coverage decreases by 94 tasks, because at most $|A|^2$ states can be selected.

Offline vs. Online Diversification

We now evaluate different time limits and compare the resulting algorithms to their offline counterparts. The top part of Table 2 confirms the result from Seipp, Keller, and Helmert (2020) that we cannot simply reduce the time for offline diversification (to 1 or 10 seconds) in order to minimize overall runtime, without sacrificing total coverage.

		1s	10s	100s	1000s	1200s	1500s
Coverage	offline	1057	1145	**1159**	1156	1150	1130
	online	1102	1136	1155	**1159**	1157	1146
Time Score	offline	**794.3**	693.4	421.0	87.0	59.4	26.0
	online	925.6	**934.7**	924.1	911.5	912.2	912.1

Table 2: Coverage and time scores for offline and online diversification using different time limits for diversification. The online variants use the *interval-10K* strategy.

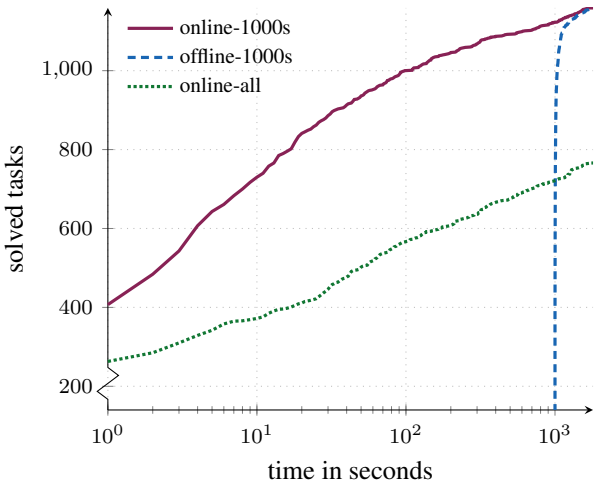

Figure 1: Number of solved tasks over time.

Offline diversification solves the highest number of tasks (1159) with a time limit T of 100 seconds and slightly fewer tasks (1156) with T=1000s. Using lower or higher time limits leads to solving much fewer tasks. The results are similar for online diversification, which solves the most tasks (1159) for T=1000s and slightly fewer tasks (1155 and 1157) for T=100s and T=1200s. Online diversification is less susceptible to the chosen time limit than offline diversification: while the difference between the maximum and minimum coverage score for offline diversification is 102 tasks, the corresponding value for online diversification is only 57 tasks.

Not only does online diversification obtain high coverage scores, but it also drastically reduces the overall runtime for many tasks compared to offline diversification. The bottom part of Table 2 reveals that all time scores of the online variants are higher than the best time score of all offline variants. The time score gap between the two variants is 131.3 points for T=1s and it grows to 886.1 points for T=1500s.

Figure 1 shows the cumulative number of solved tasks over time by offline and online diversification (with T=1000s) and the variant that computes an SCP heuristic for each evaluated state without storing any orders. The latter variant (online-all) solves the simpler tasks quickly, but only reaches a total coverage of 766 tasks. The offline variant achieves a much higher total coverage (1156 tasks), but it can only start finding solutions after its diversification phase ended. The online variant with diversification combines the advantages of the other two approaches and achieves both short runtimes and high total coverage (1159 tasks). For all time limits between 1 and 1800 seconds, online diversification solves more tasks than offline diversification and the online-all variant. Also, online-1000s solves 1122 tasks before offline-1000s even finishes the diversification phase.

Related Work

The work that is most closely related to ours simultaneously refines a set of Cartesian abstraction heuristics and a set of SCP heuristics over them during an A* search (Eifler and Fickert 2018). Whenever the maximum over the SCP heuristics violates the Bellman optimality equation (1957) for a state s and its successor states, the authors either refine one of the abstractions until the heuristic estimate for s increases, merge two abstractions or compute a new greedy order ω for s (using the h scoring function, Seipp, Keller, and Helmert 2020) and add h^{SCP}_{ω} to the set of SCP heuristics. Their strongest algorithm compares favorably against a version that only refines the abstractions offline and only computes a single SCP heuristic over them. However, both the online and the offline version are outperformed by the version that diversifies a set of SCP heuristics over a *fixed* set of Cartesian abstraction heuristics, i.e., the offline SCP variant we describe in Algorithm 2.

The literature contains additional approaches that improve heuristics online during the search. For example, the SymBA* planner repeatedly switches between a symbolic forward search and symbolic backward searches in one of multiple abstractions (Torralba, Linares López, and Borrajo 2016). In the setting of satisficing planning, Fickert and Hoffmann (2017) refine the $h^{C\text{FF}}$ heuristic, an extension of h^{FF} (Hoffmann and Nebel 2001), during enforced hill-climbing and greedy best-first searches.

As a final example, Franco and Torralba (2019) interleave the precomputation of a symbolic abstraction heuristic and the symbolic search that uses it, by iteratively switching between the two phases. In each round they double the amount of time given to each phase. Our work is orthogonal to theirs since the two approaches focus on interleaving two different types of precomputation with the search. It will be interesting to investigate whether we can also interleave the decision which abstractions to build with the search (e.g., patterns for PDBs), allowing the search to start immediately.

Conclusions

The best previously-known method for computing diverse SCP heuristics uses a fixed amount of time for sampling states and computing SCP heuristics for them. It yields strong heuristics, but needs a long precomputation phase. Computing an SCP heuristic for each evaluated state yields even better estimates and needs no precomputation phase, but it greatly slows down the search. We showed that by *diversifying* SCP heuristics *online*, we can combine the strengths of both approaches and obtain an algorithm that needs no sample states nor precomputation phase, evaluates states quickly and achieves high coverage.

Acknowledgments

We thank Malte Helmert and the anonymous reviewers for their insightful comments. We have received funding for this work from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 817639). Moreover, this research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215.

References

Bellman, R. E. 1957. *Dynamic Programming*. Princeton University Press.

Eifler, R.; and Fickert, M. 2018. Online Refinement of Cartesian Abstraction Heuristics. In *Proc. SoCS 2018*, 46–54.

Fickert, M.; and Hoffmann, J. 2017. Complete Local Search: Boosting Hill-Climbing through Online Relaxation Refinement. In *Proc. ICAPS 2017*, 107–115.

Franco, S.; and Torralba, Á. 2019. Interleaving Search and Heuristic Improvement. In *Proc. SoCS 2019*, 130–134.

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.; Bonet, B.; and Geffner, H. 2005. New Admissible Heuristics for Domain-Independent Planning. In *Proc. AAAI 2005*, 1163–1168.

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.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proc. ICAPS 2007*, 176–183.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR* 14: 253–302.

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over Multiple Pattern Databases Speeds up Heuristic Search. *AIJ* 170(16–17): 1123–1136.

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.

Katz, M.; and Domshlak, C. 2008. Optimal Additive Composition of Abstraction-based Admissible Heuristics. In *Proc. ICAPS 2008*, 174–181.

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

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *Proc. ECAI 2012*, 540–545.

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

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. 2018a. *Counterexample-guided Cartesian Abstraction Refinement and Saturated Cost Partitioning for Optimal Classical Planning*. Ph.D. thesis, University of Basel.

Seipp, J. 2018b. Fast Downward Scorpion. In *IPC-9 planner abstracts*, 77–79.

Seipp, J. 2020. Online Saturated Cost Partitioning for Classical Planning. In *ICAPS Workshop on Heuristics and Search for Domain-independent Planning*, 16–22.

Seipp, J. 2021. Code and data for the ICAPS 2021 paper "Online Saturated Cost Partitioning for Classical Planning". https://doi.org/10.5281/zenodo.4607322.

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. 2017. A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning. In *Proc. ICAPS 2017*, 259–268.

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

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New Optimization Functions for Potential Heuristics. In *Proc. ICAPS 2015*, 193–201.

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

Torralba, Á.; Linares López, C.; and Borrajo, D. 2016. Abstraction Heuristics for Symbolic Bidirectional Search. In *Proc. IJCAI 2016*, 3272–3278.