Online Saturated Cost Partitioning

Abstract

Saturated cost partitioning is a general method for admissibly adding heuristic estimates for optimal state-space search. The algorithm strongly depends on the order in which it considers the heuristics. The strongest previous approach precomputes a set of diverse orders and the corresponding saturated cost partitionings before the search. This makes evaluating the overall heuristic very fast, but requires a long precomputation phase. By diversifying the set of orders online during the search we drastically speed up the planning process and even solve slightly more tasks.

Saturated Cost Partitioning

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 (under the reduced cost functions) admissible.

Saturated cost partitioning (SCP) is one of the strongest methods for finding cost partitionings (Seipp and Helmert 2014; Seipp, Keller, and Helmert 2020). At the core of the SCP algorithm lies the insight that we can often reduce the (action) cost function of a planning task and still obtain the same heuristic estimates. This notion is captured by so-called *saturated* cost functions. An (action) 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 $\mathit{scf}(a) \leq \mathit{cost}(a)$ for each action a and 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*. We call a function that computes a saturated cost function for a given heuristic and cost function a *saturator*. The original formulation assumed S' to always be the set of all states. This definition has been generalized recently to allow preserving the estimates for a *subset* of states and we use the *perim** saturator for choosing the subsets here, because it yields the strongest heuristics (Seipp and Helmert 2019).

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

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: $\quad C \leftarrow \langle\rangle$
3: $\quad$ **for all** $h \in \omega$ **do**
4: $\quad\quad \mathit{scf}_h \leftarrow \mathit{saturate}_h(\mathit{cost})$
5: $\quad\quad C \leftarrow C \oplus \mathit{scf}_h$
6: $\quad\quad \mathit{cost} \leftarrow \mathit{cost} - \mathit{scf}_h$
7: $\quad$ **return** C

Algorithm 1 shows how the SCP procedure 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, i.e., it lets a saturator $\mathit{saturate}_h$ (which is fixed to be *perim** here) compute the fraction of the action costs that are needed to preserve the estimates by h for a subset of states under the original cost function (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 $h^{\mathrm{SCP}}_{\omega}$ for the cost partitioning heuristic that results from applying the SCP algorithm to the heuristic order ω.

The quality of a SCP heuristic greatly depends on the order in which the heuristics are considered. In this work, we use the *greedy* ordering method with the $\frac{h}{\mathit{stolen}}$ scoring function, the best ordering in previous work on saturated cost partitioning (Seipp, Keller, and Helmert 2020). As in previous work on SCP, we focus on *abstraction* heuristics (Helmert, Haslum, and Hoffmann 2007).

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. Algorithm 2 shows the strongest offline SCP algorithm from

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

1: **function** OFFLINEDIVERSIFICATION
2: $\Omega \leftarrow \emptyset$
3: $\hat{S} \leftarrow$ sample 1000 states
4: **repeat**
5: $s \leftarrow$ sample state
6: $\omega \leftarrow$ greedy order for s
7: **if** $\exists s' \in \hat{S} : h^{\text{SCP}}_{\omega}(s') > \sup_{\omega' \in \Omega} h^{\text{SCP}}_{\omega'}(s')$ **then**
8: $\Omega \leftarrow \Omega \cup \{\omega\}$
9: **until** time spent in function $\geq T$
10: **return** Ω

the literature. 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 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 high 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-nodiv*, computes a greedy order and the corresponding SCP heuristic for each state evaluated during the search. By design, online-nodiv 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-nodiv method has been shown to work well 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).

Online Diversification of SCP Heuristics

In this work, we combine ingredients of the offline and online-nodiv 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 shows pseudo-code for the approach, which adapts the COMPUTEHEURISTIC function used to evaluate

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: $\omega \leftarrow$ greedy order for s
4: **if** $h^{\text{SCP}}_{\omega}(s) > \sup_{\omega \in \Omega} h^{\text{SCP}}_{\omega}(s)$ **then**
5: $\Omega \leftarrow \Omega \cup \{\omega\}$
6: **return** $\max_{\omega \in \Omega} h^{\text{SCP}}_{\omega}(s)$

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.[1] 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).

Compared to offline diversification, this algorithm has the advantage that it allows the A* search to start immediately and it doesn't need to sample states with random walks, but can judge the utility of storing an order based on states that are actually evaluated during the search.

Time Limit

For abstraction heuristics, the offline diversification can perform two rather subtle optimizations compared to the online diversification: after precomputing all SCP heuristics, we can delete all abstract transition systems from memory, 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 precomputed orders, we can even delete the corresponding abstraction functions (Seipp 2018). While both optimizations often greatly 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.

To allow the online diversification to do these two optimizations, we need to stop the diversification eventually. We therefore introduce a time limit T and only select a state for diversification (line 2) if the total time spent in COMPUTEHEURISTIC is less than T.

State Selection Strategies

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

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

Intervals 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-nodiv), 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.

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. 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.[2]

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).

State Selection Strategies

In the first experiment, we compare the different instantiations of the SELECT function. The left and middle parts of Table 1 hold per-domain and overall coverage results for the interval strategy with different intervals and the novelty strategy for novelty 1 and 2. All strategies use a time limit of 1000 seconds for the online diversification. We see that the overall coverage is roughly the same for the interval variants (1141–1143 solved tasks) and that the higher intervals 100 and 1000 have an edge over the lower intervals 1 and 10 in per-domain coverage comparisons. The novelty variants are preferable to all interval variants in per-domain comparisons and also solve more tasks in total (1149 and 1151 tasks, respectively). These results suggest that explicitly focusing on states with previously unseen atom combinations helps to find stronger SCP heuristics and therefore we use the novelty-2 strategy in the experiments below.

[2]References removed for review.

T=1000s	interval-1	interval-10	interval-1000	interval-100	novelty-1	novelty-2	T=1000s	T=∞
interval-1	–	3	4	3	1	1	1142	802
interval-10	**6**	–	3	1	2	3	1141	941
interval-1000	**8**	**6**	–	4	2	3	1142	1049
interval-100	**7**	**3**	**5**	–	3	3	1143	1097
novelty-1	**7**	**5**	**5**	**4**	–	4	1149	1138
novelty-2	**8**	**7**	**6**	**5**	**5**	–	**1151**	1055

Table 1: Left: per-domain coverage comparisons of different state selection strategies. Each variant uses at most 1000 seconds for online diversification. The entry in row r and column c shows the number of domains in which strategy r solves more tasks than strategy c. For each strategy pair we highlight the maximum of the entries (r, c) and (c, r) in bold. Middle: total number of solved tasks with a time limit of 1000 seconds for online diversification. Right: solved tasks without a time limit.

	offline-1s	online-1s	online-10s	offline-10s	offline-1000s	online-100s	offline-100s	online-1000s	Coverage
offline-1s	–	4	1	0	0	0	0	2	1042
online-1s	**9**	–	0	2	2	0	2	2	1095
online-10s	**13**	**14**	–	4	5	1	3	4	1129
offline-10s	**12**	**13**	**5**	–	2	2	0	3	1131
offline-1000s	**15**	**16**	**12**	**8**	–	6	1	3	1143
online-100s	**16**	**18**	**10**	**10**	**7**	–	4	3	1146
offline-100s	**16**	**17**	**11**	**9**	**5**	**6**	–	**5**	1146
online-1000s	**16**	**18**	**12**	**12**	**7**	**6**	**5**	–	**1151**

Table 2: Per-domain and total coverage comparison of different SCP variants. The online variants use the novelty-2 strategy. For an explanation of the data, see Table 1.

Time Limit

The middle and right parts of Table 1 confirm 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. The coverage of the novelty-1 version only decreases by 11 tasks, because the number of states selected for diversification is limited by the number of atoms A in the planning task. For novelty-2 this bound is $|A|^2$ and the interval versions are only bounded by the number of evaluated states, which is why a time limit is more important for these variants.

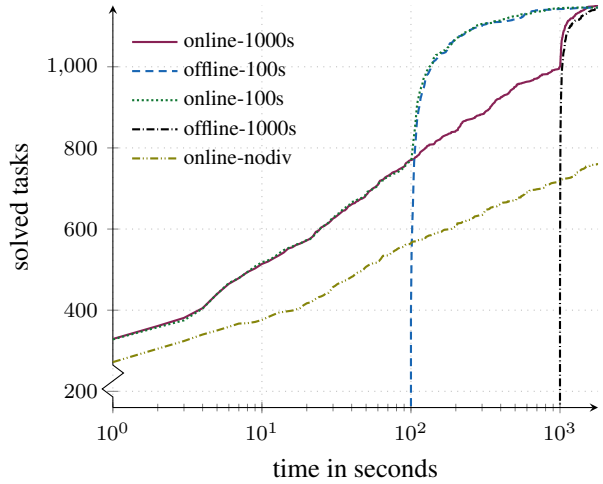

Figure 1: Number of solved tasks over time. We exclude the time for the Fast Downward preprocessing phase, which is the same for all variants.

Offline vs. Online Diversification

Having established that we need a time limit for online diversification and that the novelty-2 strategy leads to the highest coverage, we now evaluate different time limits and compare the resulting algorithms to their offline counterparts. 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 reduce overall runtime, without sacrificing total coverage. For a time limit of 100 seconds, offline diversification solves more tasks than online diversification in 6 domains, while the opposite is true in 4 domains. The picture is reversed for a time limit of 1000 seconds: here, online diversification has an edge over the offline variant in a per-domain comparison (7 vs. 3 domains). Online diversification for 1000 seconds also leads to the highest total coverage of 1151 tasks, solving 5 tasks more than the best previously-known method for finding SCP heuristics.

Not only does online diversification obtain high coverage scores, but it also drastically reduces the overall runtime for many tasks compared to offline diversification. Figure 1 shows the cumulative number of solved tasks over time by the strongest algorithms from Table 2 and the variant that computes an SCP heuristic for each evaluated state without any diversification. The latter variant (online-nodiv) solves the simpler tasks quickly, but only reaches a total coverage of 760 tasks. The offline variants that diversify for 100 and 1000 seconds achieve a much higher total coverage (1146 and 1143 tasks, respectively), but they can only start finding solutions after their diversification phase ended.

The online variants with diversification combine the advantages of the other approaches and achieve both short runtimes and high total coverage. For example, online-1000s solves 999 tasks before offline-1000s even finishes the diversification phase. After reaching the diversification time limit, the online and offline variants solve roughly the same number of additional tasks per time step.

The plot also shows that it is useful to reduce the diversification time limit from 1000 to 100 seconds if fast overall runtimes are more important than maximum coverage after 30 minutes. This holds for both the offline and online variants. For all of the four tested time limits (1s and 10s not depicted) and almost all overall time limits, however, online diversification solves more tasks than offline diversification.

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 appproaches 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 satisificing planning, Fickert and Hoffmann (2017) refine the FF heuristic (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.

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. We showed that by performing the diversification online during the search we drastically speed up the overall algorithm and even solve slightly more tasks.

Future work could evaluate whether it is beneficial to take not only the currently evaluated state, but other states from the open list, into account when judging whether a new order for SCP should be stored.

References

Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds. 2017. *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*. AAAI Press.

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

Eifler, R., and Fickert, M. 2018. Online refinement of Cartesian abstraction heuristics. In Bulitko, V., and Storandt, S., eds., *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 46–54. AAAI Press.

Fickert, M., and Hoffmann, J. 2017. Complete local search: Boosting hill-climbing through online relaxation refinement. In Barbulescu et al. (2017), 107–115.

Franco, S., and Torralba, Á. 2019. Interleaving search and heuristic improvement. In Surynek, P., and Yeoh, W., eds., *Proceedings of the 12th Annual Symposium on Combinatorial Search (SoCS 2019)*, 130–134. AAAI Press.

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.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 176–183. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 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. *Artificial Intelligence* 170(16–17):1123–1136.

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.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 540–545. IOS Press.

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 Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364. AAAI Press.

Seipp, J., and Helmert, M. 2014. Diverse and additive Cartesian abstraction heuristics. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. AAAI Press.

Seipp, J., and Helmert, M. 2018. Counterexample-guided Cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research* 62:535–577.

Seipp, J., and Helmert, M. 2019. Subset-saturated cost partitioning for optimal classical planning. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 391–400. AAAI Press.

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

Seipp, J.; Keller, T.; and Helmert, M. 2017. A comparison of cost partitioning algorithms for optimal classical planning. In Barbulescu et al. (2017), 259–268.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research* 67:129–167.

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

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.