Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning

Jendrik Seipp, Thomas Keller, Malte Helmert
University of Basel
Basel, Switzerland
{jendrik.seipp,tho.keller,malte.helmert}@unibas.ch

Abstract

In classical planning, cost partitioning is a method for admissibly combining a set of heuristic estimators by distributing operator costs among the heuristics. An optimal cost partitioning is often prohibitively expensive to compute. Saturated cost partitioning is an alternative that is much faster to compute and has been shown to offer high-quality heuristic guidance on Cartesian abstractions. However, its greedy nature makes it highly susceptible to the order in which the heuristics are considered. We show that searching in the space of orders leads to significantly better heuristic estimates than with previously considered orders. Moreover, using multiple orders leads to a heuristic that is significantly better informed than any single-order heuristic. In experiments with Cartesian abstractions, the resulting heuristic approximates the optimal cost partitioning very closely.

Introduction

A* search with an admissible heuristic is one of the most prominent methods for optimal classical planning. Because a single heuristic is often unable to capture all relevant aspects of a planning task, it is desirable to combine information from multiple heuristics. One way of doing so admissibly is to maximize over multiple heuristic estimates in each state (Holte et al. 2006). However, this method does not really combine multiple heuristics but merely selects the most informative one in each state.

Cost partitioning (CP) (Katz and Domshlak 2008; Yang et al. 2008) is a more sophisticated way of combining heuristics that often produces higher estimates than any single estimator can provide. By distributing the operator costs among the heuristics, cost partitioning allows to *sum* the heuristic estimates admissibly. An *optimal cost partitioning* (OCP) is usually too expensive to compute (e.g., Pommerening, Röger, and Helmert 2013), but multiple approximations with varying time vs. accuracy tradeoffs have been proposed, such as uniform CP (Katz and Domshlak 2007b), zero-one CP (e.g., Edelkamp 2006), and post-hoc optimization (Pommerening, Röger, and Helmert 2013).

Recently, Seipp and Helmert (2014) introduced the *saturated cost partitioning* (SCP) algorithm, which iteratively computes an abstraction, determines the minimum operator costs needed to preserve all abstract goal distances, and then repeats this process with the remaining operator costs.

Saturated cost partitioning assigns costs greedily and is therefore susceptible to the order in which the abstractions are computed. Seipp and Helmert considered two orders based on Bonet and Geffner's h^{add} heuristic (2001) as well as a random order. Their experimental evaluation was inconclusive, as none of the three orders consistently outperformed the others. In this paper, we study the problem of ordering abstractions for SCP in more depth.

Our analysis reveals that just changing the order of abstractions in SCP can make the difference between a perfect distance estimate and a blind one. To find good orders, we propose a hill climbing search in the space of all orders. The optimized orders found by hill climbing significantly improve over Seipp and Helmert's results. However, the results also indicate that it is often impossible to find a *single* order that provides good guidance across the state space: orders that are accurate on a given set of states often turn out to be poor in all others.

Maximizing over SCP heuristics for *multiple* random orders allows us to use accurate heuristics for many different states and significantly improves over single-order heuristics. This approach is similar to the one by Karpas, Katz, and Markovitch (2011), who maximize over multiple precomputed OCP heuristics instead of multiple SCP heuristics. Our method has the advantage that we never have to compute an OCP, which can be prohibitively expensive even for a single computation.

We show that our sets of SCP heuristics often contain heuristics that do not contribute any information during search. Similar to other work on heuristic subset selection (e.g., Lelis et al. 2016), we try to pick a subset of heuristics that complement each other well by actively searching for multiple *diverse* orders. Our strongest heuristic closely approximates the optimal cost partitioning and compares favorably to the state of the art for optimal classical planning.

Background

We consider optimal planning in the classical setting with a SAS$^+$-like (Bäckström and Nebel 1995) representation with non-negative operator costs. The details of the planning task representation do not matter for the technical contribution of this paper; all that matters is that we compute heuristic

estimates based on *abstractions* that are represented as (abstract) *transition systems*.

A transition system is a directed, labelled graph $\mathcal{T} = \langle S, \mathcal{O}, T, S_\star \rangle$ consisting of finitely many *states* S, a set of *operators* $\mathcal{O}$ used to label transitions in $\mathcal{T}$, a set of *transitions* $T \subseteq S \times \mathcal{O} \times S$, and a set of *goal states* $S_\star \subseteq S$. A triple $\langle s, o, s' \rangle \in T$ represents a transition from state s to state s' via operator o.

Given such a transition system and an (operator) *cost function* $c : \mathcal{O} \to \mathbb{R}$, we can define the *goal distance* $h_c^{\mathcal{T}}(s)$ of a state $s \in S$ in $\mathcal{T}$ as the shortest path from s to the nearest goal state in the weighted digraph obtained from $\mathcal{T}$ by assigning weight $c(o)$ to all transitions with label o. Note that even though we assume operator costs of planning tasks to be non-negative, we permit arbitrary real-valued costs in the context of transition systems.

The transition systems we study in this paper are induced by *abstraction functions* α (*abstractions* for short) that map states of a planning task to *abstract* states of a smaller transition system $\mathcal{T}^\alpha$. Given an operator cost function c, this defines the *abstraction heuristic* $h_c^\alpha(s) := h_c^{\mathcal{T}^\alpha}(\alpha(s))$, which is an admissible and consistent heuristic if c does not exceed the operator cost function of the planning task.

Cost Partitioning

For challenging planning tasks, using a single abstraction often does not lead to a very informative heuristic. Therefore, it can be beneficial to build multiple abstractions that focus on different aspects (e.g., Holte et al. 2006).

Cost partitioning distributes the operator costs of a task among multiple abstractions in such a way that the sum of abstraction heuristics is guaranteed to be admissible (Katz and Domshlak 2008). Following Pommerening et al. (2015), we allow negative costs for the component abstractions.

Definition 1. ***Cost partitioning.***
Let $\mathcal{A} = \langle \alpha_1, \ldots, \alpha_n \rangle$ be a tuple of abstractions of a planning task with operator cost function c. A cost partitioning *over $\mathcal{A}$ is a tuple $\mathcal{C} = \langle c_1, \ldots, c_n \rangle$ of cost functions whose sum is bounded by c: $\sum_{i=1}^n c_i(o) \leq c(o)$ for all $o \in \mathcal{O}$.*

The cost-partitioned heuristic *$h_{\mathcal{C}}^{\mathcal{A}}$ is defined as $h_{\mathcal{C}}^{\mathcal{A}}(s) := \sum_{i=1}^n h_{c_i}^{\alpha_i}(s)$. A cost partitioning $\mathcal{C}^*$ is* optimal *for state s if $h_{\mathcal{C}^*}^{\mathcal{A}}(s) \geq h_{\mathcal{C}}^{\mathcal{A}}(s)$ for all cost partitionings $\mathcal{C}$. We write $h_{\mathcal{A}}^{OCP}(s)$ for this optimal cost-partitioned heuristic value.*

Intuitively, since each abstraction yields an admissible estimate for the planning task, their sum is also admissible due to the way $\mathcal{C}$ divides the operator costs among the abstractions. Computing optimal cost partitionings can already be prohibitively expensive for abstractions of modest size (e.g., Pommerening, Röger, and Helmert 2013). One approximation that is fast to compute and often yields accurate heuristics is the *saturated cost partitioning* algorithm proposed by Seipp and Helmert (2014). It is based on the observation that high operator costs are "wasted" on a component abstraction if they do not contribute to the heuristic estimate from this abstraction.

Definition 2. ***Saturated cost function.***
Let α be an abstraction of a planning task and let c be a cost function. The saturated cost function *for α and c is the minimal cost function $\hat{c} \leq c$ that satisfies $h_{\hat{c}}^\alpha = h_c^\alpha$.*

Seipp and Helmert showed for the case of non-negative cost functions that such a minimum always exists and can be computed efficiently from the abstract transition system $\mathcal{T}^\alpha$. Their restriction to non-negative cost functions is easy to lift: the saturated cost function can be computed by setting $\hat{c}(o) = \max_{\langle s,o,s' \rangle}(h_c^{T^\alpha}(s) - h_c^{T^\alpha}(s'))$ for every operator o. For example, if an operator o only induces self-looping transitions in the abstract transition system, its saturated cost will be 0, reflecting the intuition that o contributes nothing to the solution under this abstraction.

Seipp and Helmert proposed the following iterative *saturated cost partitioning* (SCP) procedure:

1. Initialize the *remaining cost function* c to the cost function of the planning task.
2. Compute an abstraction α.
3. Let $\hat{c}$ be the saturated cost function for α and c.
4. Set $c := c - \hat{c}$ and repeat from step 2.

The procedure can terminate after computing a given number of abstractions or once no further useful abstraction heuristics can be found. The sequence of abstractions and saturated cost functions computed by the procedure then forms a cost-partitioned abstraction heuristic.

Due to the changing cost function, the order in which the SCP algorithm generates the abstractions can have a strong influence on the resulting heuristic.

Definition 3. ***Saturated cost partitioning and orders.***
Let $\mathcal{A} = \{\alpha_1, \ldots, \alpha_n\}$ be a set of abstractions of a planning task. The set of orders *of $\mathcal{A}$, denoted by $\Omega(\mathcal{A})$, consists of all permutations of $\langle \alpha_1, \ldots, \alpha_n \rangle$, i.e., all tuples of abstractions obtained by ordering $\mathcal{A}$ in any way.*

Given an order $\omega \in \Omega(\mathcal{A})$, h_ω^{SCP} is the saturated cost partitioning *heuristic for the sequence of abstractions ω, as computed by the algorithm described in the preceding text.*

The optimal saturated cost partitioning estimate *of a state s is $h_{\mathcal{A}}^{SCP*}(s) := \max_{\omega \in \Omega(\mathcal{A})} h_\omega^{SCP}(s)$. We say that $\omega \in \Omega(\mathcal{A})$ is an* optimal order *for s if $h_\omega^{SCP}(s) = h_{\mathcal{A}}^{SCP*}(s)$.*

Seipp and Helmert computed the abstractions "just-in-time", guiding the choice of abstraction functions in step 2. by the current remaining cost function. In contrast to their approach, in this paper we fix the abstractions $\mathcal{A}$ to be considered at the start and focus on finding good orders in $\Omega(\mathcal{A})$.

While our approaches work with any set of abstractions, in our experiments we consider Cartesian abstractions (Seipp and Helmert 2013) of the *landmark* and *goal* task decompositions by Seipp and Helmert (2014). The abstraction construction process is guided by the original cost function of the planning task.

Single Orders

We begin our analysis by discussing the relationship of SCP and OCP. It follows directly from Definition 1 that an optimal cost partitioning dominates all saturated cost partitionings, including optimal ones. We now show that there are cases where the dominance is strict.

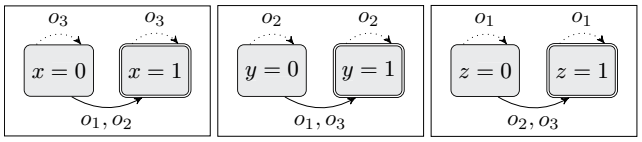

Figure 1: Abstractions for the transition system used in the proof of Theorem 1.

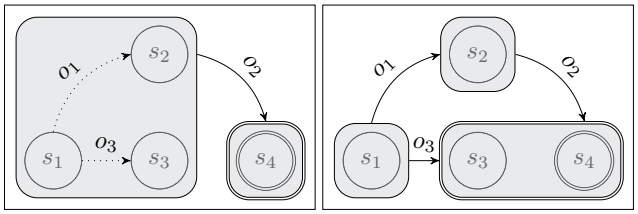

Figure 2: Abstractions α_1 (left) and α_2 (right) for the transition system used in the proof of Theorem 2. Circles depict concrete states, abstract states are rounded rectangular, and abstract self loops are dotted.

Theorem 1.
There exist planing tasks with abstractions $\mathcal{A}$ and states s such that $h^{OCP}_{\mathcal{A}}(s) > h^{SCP*}_{\mathcal{A}}(s)$.

Proof. Consider a planning task in which three binary state variables x, y and z have to be changed from 0 (in state s) to 1 (in the goal). There are three operators with cost 1 (without preconditions), each setting a different pair of variables to 1. Figure 1 shows three abstractions of the task. Regardless of the abstraction order ω, we have $h^{\text{SCP}}_{\omega}(s) = 1$. By dividing costs fractionally among the abstractions, an optimal cost partitioning yields $h^{\text{OCP}}_{\mathcal{A}}(s) = 0.5 + 0.5 + 0.5 = 1.5$. □

Note that the optimal cost partitioning for the task in Figure 1 is actually a *uniform cost partitioning* (Katz and Domshlak 2007a), so saturated cost partitioning does not in general dominate uniform cost partitioning.

The abstraction order is very important for the accuracy of the resulting SCP heuristic: two orders of the same abstractions can make the difference between a blind heuristic (estimate 0) and a perfect heuristic (estimate $h^*(s)$).

Theorem 2.
There exist planning tasks with abstractions $\mathcal{A}$ and state s such that $h^{SCP}_{\omega}(s) = h^*(s) > 0$ and $h^{SCP}_{\omega'}(s) = 0$ for two orders $\omega, \omega' \in \Omega(\mathcal{A})$.

Proof. Consider the example shown in Figure 2, where o_2 has cost 1 and o_1 and o_3 have cost 0. For the two abstractions α_1 and α_2 shown, we have $h^{\text{SCP}}_{\langle\alpha_1,\alpha_2\rangle}(s_1) = 1 = h^*(s_1)$ and $h^{\text{SCP}}_{\langle\alpha_2,\alpha_1\rangle}(s_1) = 0$. □

Note that we can arbitrarily enlarge the accuracy gap between two heuristics resulting from two different orders. In our example, it suffices to adjust the cost of o_2, and similar examples can be constructed for unit-cost problems.

Although Theorem 2 highlights the importance of choosing good orders, previous work on saturated cost partitioning only considered two orders, called $h^{\text{SCP}}_{\text{add}\uparrow}$ and $h^{\text{SCP}}_{\text{add}\downarrow}$ (Seipp and Helmert 2014). Seipp and Helmert compared these orders to a randomly selected order and observed that neither $h^{\text{SCP}}_{\text{add}\uparrow}$ nor $h^{\text{SCP}}_{\text{add}\downarrow}$ consistently outperformed the other or the random order, a result we could reproduce on a slightly larger benchmark set. As in all experiments below, we use the 1667 benchmark tasks from the optimization tracks of the 1998–2014 International Planning Competitions (IPC) and limit time and memory by 30 minutes and 2 GiB.

In our experiment, $h^{\text{SCP}}_{\text{add}\uparrow}$ and $h^{\text{SCP}}_{\text{add}\downarrow}$ solve 789 and 800 of the 1667 tasks, and selecting a random order leads to solving 759.33 tasks on average over 100 runs (with a sample standard deviation of 7.92). (Differences to the results of Seipp and Helmert are due to the altered abstraction generation.)

Even though the random orders are outperformed by both h^{add} orders, a closer look at the data reveals that better orders than $h^{\text{SCP}}_{\text{add}\uparrow}$ and $h^{\text{SCP}}_{\text{add}\downarrow}$ are fairly common: more than 25% of runs using a random order result in an SCP heuristic that requires fewer expansions to solve a task than both $h^{\text{SCP}}_{\text{add}\uparrow}$ and $h^{\text{SCP}}_{\text{add}\downarrow}$; and even though only 692 tasks are solved by all 100 random orders, 866 tasks are solved by at least one random order, significantly more (66 tasks) than $h^{\text{SCP}}_{\text{add}\downarrow}$. This suggests that finding better orders could have a positive impact on performance.

Optimized Orders

There are two challenges for finding good orders for saturated cost partitioning: first, we need to deal with a combinatorial search space of $n!$ possible orders for a set of n abstractions. Second, we are looking for orders that provide good guidance in all states visited during search and not only in a single state.

Except for very small n, it is obviously impossible to consider all $n!$ orders. We therefore address the first challenge by performing a hill climbing search in the space of orders. To deal with the second challenge, we use a set of sample states to evaluate each considered order.

We use the sampling procedure of Haslum et al. (2007) to generate a set $\hat{S}$ of 1000 sample states, using $h^{\text{SCP}}_{\text{add}\downarrow}$ to estimate the plan cost and to exclude states s with $h^{\text{SCP}}_{\text{add}\downarrow}(s) = \infty$ from the sample set. We start the hill climbing search with a random incumbent order ω and assess its quality as the sum of heuristic values on the samples, i.e., $q(\omega) = \sum_{s\in\hat{S}} h^{\text{SCP}}_{\omega}(s)$. Afterwards, we generate the set of neighboring orders by switching any two positions in the incumbent order. This *two-exchange* neighborhood is common for local search optimization algorithms (Pisinger and Ropke 2010) and guarantees that all orders can be reached from any initial order. The first neighbor ω' with $q(\omega') > q(\omega)$ becomes the new incumbent. We repeat this procedure until no neighbor improves on the incumbent. The final incumbent is then used as the basis of our SCP heuristic, which we call $h^{\text{SCP}}_{\text{HC}}$.

The fourth column in Table 1 shows results from 10 $h^{\text{SCP}}_{\text{HC}}$ runs with different random seeds. On average, $h^{\text{SCP}}_{\text{HC}}$ solves 884.9 tasks with a sample standard deviation of 4.7, amount-

Coverage	$h^{\text{SCP}}_{\text{rand1}}$	$h^{\text{SCP}}_{\text{add}\uparrow}$	$h^{\text{SCP}}_{\text{add}\downarrow}$	$h^{\text{SCP}}_{\text{HC}}$	$h^{\text{SCP}}_{\text{rand200}}$	$h^{\text{SCP}}_{\text{div}}$	$h^{\text{SEQ+}}_{\text{LM-cut}}$
airport (50)	23.4	23	24	25.0	25.0	25.0	**29**
barman (34)	**4.0**	**4**	**4**	**4.0**	**4.0**	**4.0**	**4**
blocks (35)	18.7	20	18	24.5	23.6	26.7	**29**
childsnack (20)	**0.0**	**0**	**0**	**0.0**	**0.0**	**0.0**	**0**
depot (22)	6.6	7	6	**11.0**	**11.0**	**11.0**	7
driverlog (20)	10.8	**14**	10	13.9	**14.0**	**14.0**	13
elevators (50)	33.6	35	31	38.0	**44.0**	**44.0**	35
floortile (40)	2.0	2	2	2.0	2.0	2.0	**11**
freecell (80)	35.8	43	**68**	**68.0**	64.7	67.5	31
ged (20)	**15.0**	**15**	**15**	**15.0**	**15.0**	**15.0**	13
grid (5)	2.3	**3**	2	**3.0**	**3.0**	**3.0**	2
gripper (20)	**7.0**	**7**	**7**	**7.0**	**7.0**	**7.0**	6
hiking (20)	11.5	11	12	12.7	**13.0**	**13.0**	8
logistics (63)	26.6	27	28	30.4	38.8	**39.0**	26
miconic (150)	75.7	70	70	98.3	**144.0**	**144.0**	141
movie (30)	**30.0**	**30**	**30**	**30.0**	**30.0**	**30.0**	**30**
mprime (35)	25.2	25	**26**	**26.0**	**26.0**	**26.0**	22
mystery (30)	**17.0**	**17**	**17**	**17.0**	**17.0**	**17.0**	16
nomystery (20)	15.3	14	14	19.9	**20.0**	**20.0**	12
openstacks (100)	45.0	**47**	45	45.0	45.0	45.0	35
parcprinter (50)	22.4	30	38	36.9	35.6	38.1	**49**
parking (40)	0.2	0	0	5.7	6.3	**7.0**	5
pathways (30)	4.0	4	4	4.0	4.0	4.0	**5**
pegsol (50)	45.0	46	46	46.0	46.0	**48.0**	46
pipes-nt (50)	17.3	19	18	20.2	22.0	**22.5**	14
pipes-t (50)	12.9	14	14	14.0	14.0	**15.9**	10
psr-small (50)	49.0	49	49	49.0	49.0	49.0	**50**
rovers (40)	**7.0**	**7**	**7**	**7.0**	**7.0**	**7.0**	**7**
satellite (36)	6.0	6	6	6.0	**7.0**	**7.0**	**7**
scanalyzer (50)	22.1	**23**	21	**23.0**	**23.0**	**23.0**	**23**
sokoban (50)	41.4	43	41	43.2	42.9	44.0	**49**
storage (30)	**16.0**	**16**	**16**	**16.0**	**16.0**	**16.0**	15
tetris (17)	7.0	7	7	7.0	7.0	7.0	**11**
tidybot (40)	**22.0**	**22**	**22**	**22.0**	**22.0**	**22.0**	10
tpp (30)	6.2	6	6	7.0	**8.0**	**8.0**	**8**
transport (70)	23.0	23	23	23.0	**24.0**	**24.0**	23
trucks (30)	9.6	**12**	**12**	**12.0**	**12.0**	**12.0**	10
visitall (40)	12.1	13	12	13.0	14.0	14.0	**34**
woodwork (50)	18.7	23	17	27.2	27.1	32.0	**37**
zenotravel (20)	12.0	12	12	12.0	**13.0**	**13.0**	12
Sum (1667)	759.3	789	800	884.9	947.0	**966.7**	895
Stddev.	7.92	–	–	4.7	1.7	0.82	–

Table 1: Number of solved tasks. Results for randomized configurations are averaged over 10 runs.

ing to 95.9 and 84.9 more tasks than $h^{\text{SCP}}_{\text{add}\uparrow}$ and $h^{\text{SCP}}_{\text{add}\downarrow}$. This is a huge improvement in the optimal classical planning setting, where task difficulty tends to scale exponentially.

Furthermore, almost all tasks solved by $h^{\text{SCP}}_{\text{add}\uparrow}$, $h^{\text{SCP}}_{\text{add}\downarrow}$ or any of the 100 random orders are also solved by at least one $h^{\text{SCP}}_{\text{HC}}$ run, with only 3 counterexamples. These results show that our optimization procedure is able to reliably find high-quality orders.

Multiple Orders

So far, we have focused on finding a single order that provides good guidance for all states encountered during search. However, such an order does not always exist.

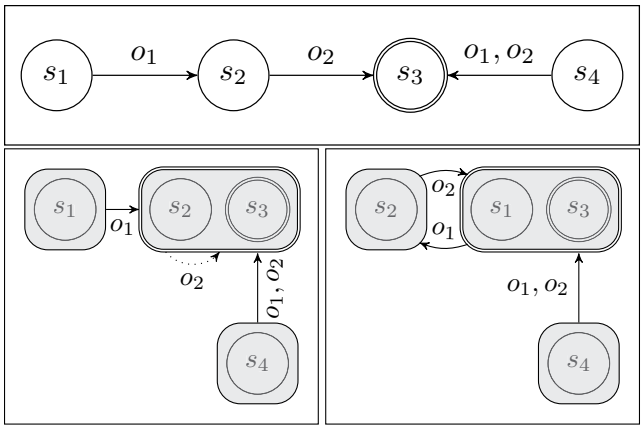

Figure 3: Concrete transition system $\mathcal{T}$ (above) with abstractions α_1 (bottom left) and α_2 (bottom right) used in the proof of Theorem 3. Circles depict concrete states, abstract states are rounded rectangular, and abstract self loops are dotted.

Orders	1	2	5	10	100	200	500	1000
Coverage	759.3	797.3	866.3	907.5	942.8	**947.0**	936.3	879.5
Stddev.	7.92	6.34	2.79	4.01	2.3	1.7	1.42	3.03

Table 2: Number of solved tasks when maximizing over multiple orders. Results are averaged over 10 runs.

Theorem 3.
There exist planning tasks with abstractions $\mathcal{A}$ and states s, s' such that $h^{SCP}_{\omega}(s) > 0$, $h^{SCP}_{\omega'}(s') > 0$, and $h^{SCP}_{\omega''}(s) = h^{SCP}_{\omega'''}(s') = 0$ for two orders ω and ω' and all orders $\omega'' \neq \omega$ and $\omega''' \neq \omega'$ of $\mathcal{A}$.

Proof. Consider the example task and corresponding abstractions α_1 and α_2 in Figure 3. All operators cost 1. We have $h^{\text{SCP}}_{\langle\alpha_1,\alpha_2\rangle}(s_1) = 1$, $h^{\text{SCP}}_{\langle\alpha_2,\alpha_1\rangle}(s_1) = 0$, $h^{\text{SCP}}_{\langle\alpha_1,\alpha_2\rangle}(s_2) = 0$, and $h^{\text{SCP}}_{\langle\alpha_2,\alpha_1\rangle}(s_2) = 1$. □

Theorem 3 can be generalized to an arbitrary number of states and abstractions. Unfortunately, this means that there are planning tasks where all orders lead to heuristics that degenerate to the blind heuristic on all but a few states, and hence there is no single order that gives good guidance across the board. Computing a good order in *every* evaluated state overcomes this dilemma, but the increased heuristic accuracy usually does not compensate for the additional computation time (e.g., Karpas, Katz, and Markovitch 2011; Seipp, Pommerening, and Helmert 2015).

Instead, we pursue an alternative with a good tradeoff between heuristic accuracy and computation time by generating heuristics for *multiple* orders and using the maximum over their estimates in each state. Table 2 shows the number of solved tasks when maximizing over a set of random orders Ω for an increasing size of Ω. With up to 200 random orders, the number of solved tasks increases steadily and reaches the maximum with a coverage of 947.0 on average

– an improvement of 187.7 tasks compared to the configuration that uses only a single random order, of 147.0 compared to the best order of Seipp and Helmert (2014) and of 62.1 in comparison to $h^{\text{SCP}}_{\text{HC}}$.

Unfortunately, this approach does not turn out to be complementary to the hill-climbing optimization approach described in the previous section. Using multiple optimized orders instead of random ones did not increase performance on the evaluated benchmarks. A possible explanation is that most random orders are well-suited to guide an A* search in at least some states and a sufficiently large set of random orders covers all (or at least sufficiently many) evaluated states in this way.

The same hypothesis could explain the fact that using more than 200 orders leads to fewer solved tasks: we know that adding an order to an existing set of orders can only increase the accuracy of the resulting heuristic. Since coverage decreases when using more than 200 random orders, the gain in accuracy from including additional orders must be outweighed by the increased computational and memory cost. Since these costs are modest, this suggests that the gain in accuracy becomes negligible after a certain point.

To test this hypothesis, we measure the number of individual orders that actually contribute to the overall heuristic. For this purpose, we keep track of the sets of orders that induce the highest heuristic estimates for each encountered state. We say that all orders in the *minimal hitting set*[1] of these sets are *useful* for the search, and all others are *useless*. The intuition behind this definition is that a search with a heuristic that discards all heuristics based on useless orders evaluates exactly the same states as a search with all heuristics.

As the computation of a minimum hitting set is NP-complete (Karp 1972), we approximate it by using a greedy algorithm that treats the set of orders Ω as a sequence $\langle \omega_1, \ldots, \omega_n \rangle$ (any order suffices). Then an order ω_i is *probably useful* for state $s \in S$ if it is the first order in the sequence that maximizes the heuristic value, i.e., $h^{\text{SCP}}_{\omega_i}(s) > h^{\text{SCP}}_{\omega_j}(s)$ for all $j < i$, and $h^{\text{SCP}}_{\omega_i}(s) \geq h^{\text{SCP}}_{\omega_j}(s)$ for all $j > i$. We call all orders that are probably useful for at least one state encountered during search *probably useful*. The set of probably useful orders is a hitting set, but not necessarily a minimal one. It therefore serves as an upper bound on the number of orders that contribute to the search.

Figure 4 compares the percentage of probably useful orders for two different heuristics. For the moment, we are only interested in the $h^{\text{SCP}}_{\text{rand200}}$ configuration on the x-axis, which shows the percentage of probably useful orders in Ω with $|\Omega| = 200$. There are only a few planning tasks where more than 40% of the orders are probably useful, and many where less than 20% of the orders contribute to the search. If we take into account that these numbers are an upper bound on the real number of useful orders, we can confirm that it is rarely useful to add another random order to an already large set of orders.

[1]We can break ties arbitrarily, so for the sake of simplicity, we assume that there is exactly one minimal hitting set.

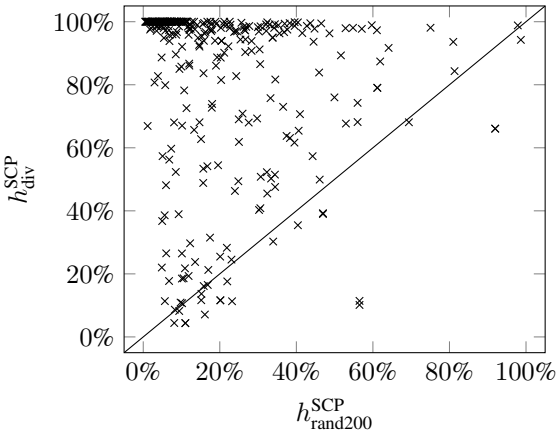

Figure 4: Percentage of *probably useful* orders for 200 random orders ($h^{\text{SCP}}_{\text{rand200}}$) and diverse orders ($h^{\text{SCP}}_{\text{div}}$). We exclude tasks for which any of the two heuristics uses fewer than 1000 expansions. Results are averaged over 10 runs.

Time	0.1s	1s	10s	100s	200s	500s	1000s	1200s
Coverage	928.3	948.0	962.3	965.3	**966.7**	966.5	962.3	956.3
Stddev.	2.79	2.05	0.95	1.34	0.82	0.97	0.95	1.25

Table 3: Number of solved tasks by $h^{\text{SCP}}_{\text{div}}$ for different amounts of time for finding orders. Results are averaged over 10 runs.

Diverse Orders

The analysis of probably useful orders not only explains why additional orders lead to a lower coverage once Ω reaches a certain size, but it also shows that the convincing results of $h^{\text{SCP}}_{\text{rand200}}$ are obtained *despite* having a large number of useless orders in Ω. Removing the useless orders from Ω would result in faster heuristic evaluation without loss of information, and replacing them with useful orders would result in a more accurate heuristic.

Unfortunately, we can only decide whether an order is useful once the search has terminated. We therefore sample a set $\hat{S}$ of 1000 sample states and use it as a proxy for the real set of states that is encountered during the search.

We propose the following diversification algorithm for finding a set of useful orders Ω: first, we initialize Ω to be the empty set. Afterwards, until a given time limit T is reached, we iteratively generate a random order ω, add it to Ω if $h^{\text{SCP}}_{\omega}(s) > \max_{\omega' \in \Omega} h^{\text{SCP}}_{\omega'}(s)$ for at least one state $s \in \hat{S}$, and discard it otherwise.

Table 3 shows the total number of tasks solved by the resulting heuristic $h^{\text{SCP}}_{\text{div}}$ for various time limits T used for diversification. The algorithm solves more tasks with increasing T until it reaches its peak at $T = 200$ seconds. At the peak, $h^{\text{SCP}}_{\text{div}}$ has a total average coverage of 966.7 tasks, solves at least as many tasks as $h^{\text{SCP}}_{\text{rand200}}$ in every domain and has a higher coverage than $h^{\text{SCP}}_{\text{rand200}}$ in 10 domains. It furthermore outperforms all single-order SCP techniques signifi-

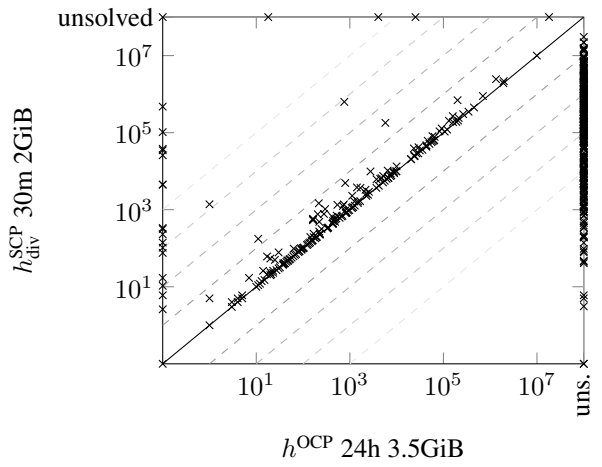

Figure 5: Number of expansions before the last f layer for h^{OCP} and $h^{\text{SCP}}_{\text{div}}$. Results for $h^{\text{SCP}}_{\text{div}}$ are averaged over 10 runs.

cantly, and solves another 19.7 tasks on average compared to $h^{\text{SCP}}_{\text{rand200}}$. $h^{\text{SCP}}_{\text{div}}$ therefore refers to the version with $T = 200$ seconds below.

We believe that one of the reasons for the increase in coverage is that the selected orders are more diverse, and hence there are fewer relevant states where the heuristic guidance of $h^{\text{SCP}}_{\text{div}}$ is poor. This is true even though the average size of Ω is slightly lower for $h^{\text{SCP}}_{\text{div}}$ (81.26 orders when using the time limit of 200 seconds, compared to 200 for $h^{\text{SCP}}_{\text{rand200}}$). On some tasks, only a single order is useful, and on some more than 1000 useful orders are detected during diversification. Even though the percentage of useful orders can be expected to be larger if Ω is smaller, the difference does not make up for the vastly superior impression that can be seen in Figure 4: the percentage of probably useful orders of $h^{\text{SCP}}_{\text{div}}$ is higher than for $h^{\text{SCP}}_{\text{rand200}}$ in almost all tasks. Moreover, most of the data points show that more than 60% of the orders of $h^{\text{SCP}}_{\text{div}}$ are probably useful, and there is even a significant amount of tasks where almost all orders of $h^{\text{SCP}}_{\text{div}}$ improve the overall heuristic in at least one relevant state.

Comparison to Other Approaches

The preceding experiments have shown three significant improvements in quality for SCP-based heuristics: first, by optimizing the abstraction order; second, by considering multiple orders; and finally, by explicitly searching for diversity among orders. This naturally raises the question how close our best configuration approximates the heuristic quality of an optimal cost partitioning.

Figure 5 compares the number of expansions of h^{OCP} and $h^{\text{SCP}}_{\text{div}}$. For almost all commonly solved tasks, $h^{\text{SCP}}_{\text{div}}$ is about as accurate as h^{OCP}. However, since $h^{\text{SCP}}_{\text{div}}$ is much faster to evaluate than h^{OCP}, $h^{\text{SCP}}_{\text{div}}$ solves significantly more tasks in 30 minutes using at most 2 GiB (966.7 tasks) than h^{OCP} solves in 24 hours using at most 3.5 GiB (531 tasks).

We also compare our SCP heuristics to some state-of-the-art admissible heuristics from the literature, namely the operator counting heuristic with the state equation and LM-Cut constraints ($h^{\text{SEQ+}}_{\text{LM-cut}}$) (Pommerening et al. 2014), $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009), h^{iPDB} (Haslum et al. 2007), the diverse potentials heuristic ($h^{\text{pot}}_{\text{div}}$) (Seipp, Pommerening, and Helmert 2015), merge-and-shrink using bisimulation and the DFP merge strategy ($h^{\text{M\&S}}$) (Helmert et al. 2014; Sievers, Wehrle, and Helmert 2014), and the state-equation heuristic (h^{SEQ}) (Bonet 2013). The approach by Karpas, Katz, and Markovitch (2011), who describe the method that comes closest to ours, is inapplicable in our setting, as there are already 211 tasks solved by $h^{\text{SCP}}_{\text{div}}$ for which computing even a single optimal cost partitioning takes more than 30 minutes.

	$h^{\text{SCP}}_{\text{div}}$	$h^{\text{SEQ+}}_{\text{LM-cut}}$	$h^{\text{LM-cut}}$	$h^{\text{iPDB}}_{\text{900s}}$	$h^{\text{pot}}_{\text{div}}$	$h^{\text{M\&S}}$	h^{SEQ}
Coverage	**966.7**	895	882	814	804	743	734
#Dom. $h^{\text{SCP}}_{\text{div}}$ better	–	23	20	18	24	25	27
#Dom. $h^{\text{SCP}}_{\text{div}}$ worse	–	10	10	9	8	6	7

Table 4: Overall and per-domain comparison of $h^{\text{SCP}}_{\text{div}}$ to the strongest Fast Downward heuristics.

Due to space reasons, Table 3 lists per-domain coverage results only for $h^{\text{SEQ+}}_{\text{LM-cut}}$, which has the highest total coverage among the mentioned heuristic from the literature. We summarize the results for the other heuristics in Table 4. The $h^{\text{SCP}}_{\text{div}}$ heuristic significantly outperforms all other heuristics, solving 71.7 more tasks than the best of them ($h^{\text{SEQ+}}_{\text{LM-cut}}$).

Beyond total coverage, $h^{\text{SCP}}_{\text{div}}$ also outperforms the other heuristics in most individual domains. Table 4 shows that out of the 40 tested domains, $h^{\text{SCP}}_{\text{div}}$ solves more tasks than $h^{\text{LM-cut}}$ in 20 domains, while the opposite is true in only 10 domains.

This significant increase in solved tasks compared to other admissible heuristics raises the question how $h^{\text{SCP}}_{\text{div}}$ compares to the winner of the IPC 2014 sequential optimization track, the symbolic search planner SymBA$^*_2$ (Torralba, Linares López, and Borrajo 2016). To allow for an unbiased comparison, we evaluate a version of $h^{\text{SCP}}_{\text{div}}$ that uses h^2 mutexes to prune irrelevant operators (Alcázar and Torralba 2015), a preprocessing technique that is an important ingredient of SymBA$^*_2$ and can be combined with any planning algorithm. We compare it to a version of SymBA$^*_2$ that differs from the IPC version only in some bug-fixes. Both planners perform equally in a per-domain comparison, with each having an edge over the other in 17 domains. In terms of total coverage, our $h^{\text{SCP}}_{\text{div}}$-based planner has a very slight edge, solving 1017.4 tasks compared to 1008 for SymBA$^*_2$.

Conclusion

We showed both theoretically and in experiments that the order in which saturated cost partitioning considers a set of abstractions greatly influences the quality of the resulting heuristic.

Orders optimized by a hill-climbing search result in significantly more accurate heuristics than those obtained with previously proposed orders. Maximizing over heuristics

from multiple orders leads to further improvements, especially when explicitly diversifying the set of orders to include only those that prove useful on a set of sample states.

Our best heuristic based on saturated cost partitioning solves significantly more tasks than other state-of-the-art heuristics for explicit-state search and is also competitive with the currently strongest symbolic-search planner.

In future work, we would like to investigate the relationship between SCP and other cost partitioning algorithms besides optimal cost partitioning.

Acknowledgments

We would like to thank Álvaro Torralba for providing us with the fixed SymBA$^*_2$ version.

This work was supported by the European Research Council as part of the project "State Space Exploration: Principles, Algorithms and Applications".

References

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

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

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

Bonet, B. 2013. An admissible heuristic for SAS$^+$ planning obtained from the state equation. In *Proc. IJCAI 2013*, 2268–2274.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In *Proc. MoChArt 2006*, 35–50.

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., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *JACM* 61(3):16:1–63.

Holte, R.; 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.

Karp, R. M. 1972. Reducibility among combinatorial problems. In Miller, R. E., and Thatcher, J. W., eds., *Complexity of Computer Computations*. Plenum Press. 85–103.

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. 2007a. Structural patterns heuristics: Basic idea and concrete instance. In *ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning*.

Katz, M., and Domshlak, C. 2007b. Structural patterns of tractable sequentially-optimal planning. In *Proc. ICAPS 2007*, 200–207.

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

Lelis, L. H. S.; Franco, S.; Abisrror, M.; Barley, M.; Zilles, S.; and Holte, R. C. 2016. Heuristic subset selection in classical planning. In *Proc. IJCAI 2016*, 3185–3191.

Pisinger, D., and Ropke, S. 2010. Large neighborhood search. In Gendreau, M., and Potvin, J.-Y., eds., *Handbook of Metaheuristics*. Springer. 399–419.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *Proc. ICAPS 2014*, 226–234.

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

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

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In *Proc. ICAPS 2013*, 347–351.

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

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In *Proc. ICAPS 2015*, 193–201.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proc. AAAI 2014*, 2358–2366.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In *Proc. IJCAI 2016*, 3272–3278.

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