Representing Perfect Saturated Cost Partitioning Heuristics in Classical Planning

Paul Höft[1] , **David Speck**[2] , **Jendrik Seipp**[1]
[1]Linköping University, Sweden
[2]University of Basel, Switzerland
paul.hoft@liu.se, davidjakob.speck@unibas.ch, jendrik.seipp@liu.se

Abstract

Saturated cost partitioning (SCP) is one of the strongest methods for admissibly combining heuristics for optimal classical planning. The quality of an SCP heuristic depends heavily on the order in which its component heuristics are considered. For high accuracy, it is essential to maximize over multiple SCP heuristics computed using different component orders. However, for n component heuristics, even enumerating all $n!$ orders is usually infeasible. Consequently, previous work resorted to using greedy algorithms and local optimization. In contrast, we present the first practical method for computing the perfect SCP heuristic that is equivalent to considering all component orders. We show that a set of SCP heuristics forms an *additive disjunctive heuristic*, which allows us to concisely represent component orders as a directed acyclic graph. Furthermore, once certain components have been considered, the order of the remaining components often becomes irrelevant. By exploiting this characteristic, we can reduce the size of the heuristic representation by several orders of magnitude in practice. Finally, our work makes it possible to compare the quality of existing SCP methods with that of the perfect SCP heuristic, revealing that existing approximations are nearly optimal for standard benchmarks.

1 Introduction

The A* algorithm (Hart, Nilsson, and Raphael 1968), together with an admissible heuristic, is the most widely used approach for solving classical planning problems optimally (e.g., Haslum et al. 2007; Helmert and Domshlak 2009; Karpas and Domshlak 2009; Sievers and Helmert 2021). Today's strongest such heuristics use some form of *cost partitioning*, which admissibly combines a set of component heuristics by dividing the cost of each action among them (Katz and Domshlak 2010). One such algorithm is *saturated cost partitioning* (SCP) (Seipp and Helmert 2014; Seipp, Keller, and Helmert 2020). Given an ordered sequence of component heuristics, SCP greedily assigns as little cost as necessary to each component heuristic so that all estimates are preserved, before using the remaining costs for the next heuristic in the same way, until all heuristics have been considered.

Computing an SCP is much more efficient than *optimal cost partitioning* (Katz and Domshlak 2010; Pommerening et al. 2015; Pommerening et al. 2021) and usually more informative than *(saturated) post-hoc optimization* (Pommerening, Röger, and Helmert 2013; Höft, Speck, and Seipp 2023; Höft et al. 2024) and *(opportunistic) uniform cost partitioning* (Seipp, Keller, and Helmert 2017a). SCP has been considered both for abstraction heuristics (Rovner, Sievers, and Helmert 2019; Seipp 2019; Kreft et al. 2023; Sievers, Keller, and Röger 2024) and landmark heuristics (Seipp, Keller, and Helmert 2017a). In this paper, we focus on SCP over abstraction heuristics, the basis of the winning entries in the International Planning Competition (IPC) 2023 (Drexler et al. 2023; Seipp 2023; Taitler et al. 2024).

One drawback of SCP is that for most tasks, it is impossible to find a single order that provides accurate estimates across all evaluated states. Previous work therefore maximized over a diverse set of SCP heuristics, optimized for different states (Seipp 2017). While this approach works well in practice, it has three fundamental limitations. First, each order is greedily initialized for a sampled state s and then optimized with local search, which often fails to find the best order for s. Second, even an optimal order for s may be arbitrarily bad for other states (Seipp, Keller, and Helmert 2017b). Third, due to the factorial number of orders, it is infeasible to judge the quality of the found orders against the best possible orders.

To address these issues, we introduce an algorithm that efficiently computes all orders for SCP. Its efficiency stems from several key insights. (1) Common prefixes between orders lead to duplicate computations and lookup tables that can be avoided by representing saturated cost partitioning as an *additive disjunctive heuristic graph* (ADHG) (Coles et al. 2008). We introduce the first general reduction rules for ADHGs to reduce their memory requirements. (2) To deal with the factorial number of orders, we identify conditions under which the relative order of heuristics becomes irrelevant, allowing for the collapse of all such equivalent suborders into one. (3) We demonstrate how this *order independence* can be approximated during the construction of the ADHG to avoid generating equivalent suborders. Together, these insights allow us to efficiently compute h^{SCP}_*, the perfect SCP heuristic.

We evaluate our approach on IPC benchmarks and show that the ADHG allows us to compute h^{SCP}_* for many more tasks than the explicit alternative which enumerates all orders. We also show that the best sampling-based strategies approximate h^{SCP}_* very closely, even though they consider only a tiny fraction of orders.

2 Background

Optimal classical planning tasks can be solved with a state space search in a weighted transition system defined as $\mathcal{T} = \langle S, L, T, \mathit{cost}, s_0, S_* \rangle$ where S is a finite set of *states*, L is a finite set of *labels* for a set of *transitions* T of the form $s \xrightarrow{\ell} s'$ with $s, s' \in S$, $\ell \in L$, and a *cost function* $\mathit{cost} : L \to \mathbb{R} \cup \{-\infty, \infty\}$. The state $s_0 \in S$ defines the *initial state* and $S_* \subseteq S$ is a set of *goal states*. Solving the planning task implies finding a sequence of transitions, a *plan*, that leads from the initial state to one of the goal states. The *cost* of a plan is defined as the sum of its transition costs. In optimal planning, the task is to either find a plan with minimum cost or show that no plan exists. A label ℓ *affects* a transition system $\mathcal{T}$ if there exists a transition $s \xrightarrow{\ell} s'$ where $s \neq s'$.

A *heuristic* is a function $h : S \to \mathbb{R} \cup \{-\infty, \infty\}$ that estimates goal distances. It is admissible if $h(s) \leq h^*(s)$ for all $s \in S$, where h^* is the *perfect heuristic*, that maps each state to the cost of the cheapest path from s to any goal state. We write $h(\mathit{cost}, s)$ to emphasize that heuristic h is computed under cost function *cost*. Two heuristics h_1 and h_2 are *equivalent* if $h_1(s) = h_2(s)$ for all $s \in S$.

Abstraction heuristics use an *abstraction function* $\alpha : S \to S^\alpha$ to induce the *abstract transition system* $\mathcal{T}^\alpha = \langle S^\alpha, L, T^\alpha, \mathit{cost}, s_0^\alpha, S_*^\alpha \rangle$, with $T^\alpha = \{\alpha(s) \xrightarrow{\ell} \alpha(s') \mid s \xrightarrow{\ell} s' \in T\}$, $s_0^\alpha = \alpha(s_0)$ and $S_*^\alpha = \{\alpha(s) | s \in S_*\}$. $\mathcal{T}^\alpha$ gives rise to the admissible abstraction heuristic h^α which returns the perfect goal distance in the abstraction: $h^\alpha(s) = h^*_{\mathcal{T}^\alpha}(\alpha(s))$. To evaluate an abstraction heuristic during a search, it suffices to store the abstraction function and a *lookup table* that holds the goal distances of all abstract states.

A *cost function mapping* $\mathcal{C}$ that maps each heuristic $h_i \in \mathcal{H}$ to a cost function cost_i with $1 \leq i \leq n$ is a *cost partition*, if the *cost partitioning condition* $\sum_{i=1}^n \mathit{cost}_i(\ell) \leq \mathit{cost}(\ell)$ holds for all $\ell \in L$. The resulting *cost partitioning heuristic* $h^{\mathcal{C}} = \sum_{i=1}^n h_i(\mathit{cost}_i, s)$ is admissible if all h_i are admissible (Katz and Domshlak 2010; Pommerening et al. 2015). *Saturated cost partitioning* (SCP) considers heuristics in sequence and uses the fact that a given heuristic can often be evaluated under a reduced (*saturated*) cost function without affecting any of its estimates, allowing SCP to preserve unneeded costs for subsequent heuristics (Seipp, Keller, and Helmert 2020). Formally, for a given order $\omega = \langle h_1, \ldots, h_n \rangle$, SCP computes the cost partition $\mathcal{C}_\omega^{\text{SCP}} = \{h_i \mapsto \mathit{cost}_i\}$ with $\mathit{cost}_i = \mathit{saturate}(h_i, \mathit{remain}_{i-1})$, $\mathit{remain}_i = \mathit{remain}_{i-1} - \mathit{cost}_i$ for all $1 \leq i \leq n$ and $\mathit{remain}_0 = \mathit{cost}$. Infinite values are handled by the rules of *left-addition*, i.e., $\infty - \infty = \infty$. We write $h_\omega^{\text{SCP}} = h^{\mathcal{C}_\omega^{\text{SCP}}}$ for the resulting SCP heuristic. For abstraction heuristics h^α, we can compute the unique *minimum saturated cost function* $\mathit{mscf} = \mathit{saturate}(h^\alpha, \mathit{cost})$ with minimal overhead (Seipp and Helmert 2018):

$$\mathit{mscf}(\ell) = \sup_{s \xrightarrow{\ell} s' \in T} (h^*_{\mathcal{T}^\alpha}(\mathit{cost}, s) \ominus h^*_{\mathcal{T}^\alpha}(\mathit{cost}, s'))$$

Here, $\ominus$ denotes regular subtraction, except that $\infty - \infty = -\infty$.

3 SCP Heuristics as Computational Graphs

The accuracy of a saturated cost partitioning heuristic over abstraction heuristics $\mathcal{H}$ strongly depends on the order ω in which the SCP algorithm considers the heuristics $h \in \mathcal{H}$ (Seipp, Keller, and Helmert 2017b). It is therefore beneficial to maximize over multiple SCP heuristics h_ω^{SCP} computed for different orders ω (Seipp, Keller, and Helmert 2020). In principle, we can obtain the *perfect SCP heuristic*, h_*^{SCP}, by maximizing over *all* orders ω of $\mathcal{H}$. However, computing h_*^{SCP} naively in this way for $n = |\mathcal{H}|$ heuristics requires enumerating $n!$ orders and storing n lookup tables for each order. This factorial explosion makes the naive computation of h_*^{SCP} infeasible even for small values of n.

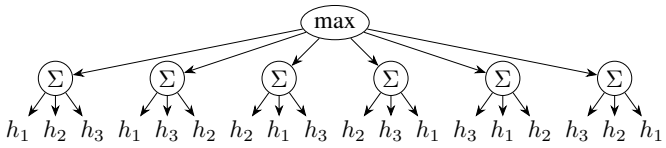
As a consequence, previous work greedily approximates good orders for single states s, by ordering those heuristics h first with a high ratio of $h(s)$ divided by the sum of saturated costs that h wants to take away from other heuristics. Greedy orders can be optimized further with local search (Seipp, Keller, and Helmert 2017b). To "cover" multiple states, Seipp, Keller, and Helmert (2020) proposed a diversification procedure that iteratively generates new SCP heuristics h_ω^{SCP} for (optimized) greedy orders ω but only keeps h_ω^{SCP} if it increases the overall heuristic estimate for at least one sample state.

Instead of approximating h_*^{SCP}, we present the first practically feasible approach to compute h_*^{SCP} exactly. While our algorithm considers all orders, it only does so *implicitly*, which significantly reduces the number of computed and stored lookup tables. The main reductions come from two advancements. First, we share the computations and lookup tables between orders that have the same prefix. And second, we approximate equivalence between orders and enumerate only those orders that can lead to different SCP heuristics. We now first show the benefits of expressing a saturated cost partitioning heuristic as a graph over its saturated component heuristics. Later, we will explore equivalence between orders in Section 4.

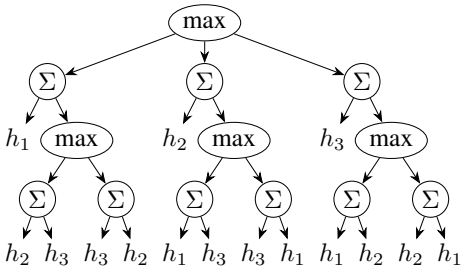
Computing and storing each order individually in an SCP heuristic is wasteful when a prefix is repeated. Two orders that share a prefix yield the same lookup tables for all heuristics in the prefix. When computing all possible orders, this drastically reduces computational effort and memory consumption. To represent saturated cost partitioning heuristics over multiple orders while sharing lookup tables between orders, we use the following graph data structure from Coles et al. (2008).

Definition 1 (Additive-Disjunctive Heuristic Graph)**.** *An additive-disjunctive heuristic graph (ADHG) is a directed acyclic graph with a single root node and three types of nodes:* sum*,* max *and* evaluator *nodes. Sum and max nodes are internal nodes and evaluate to the sum and the maximum of the heuristic values of their children, respectively. Evaluator nodes are leaf nodes and evaluate an assigned heuristic. The evaluation of the root node is the ADHG heuristic value.*

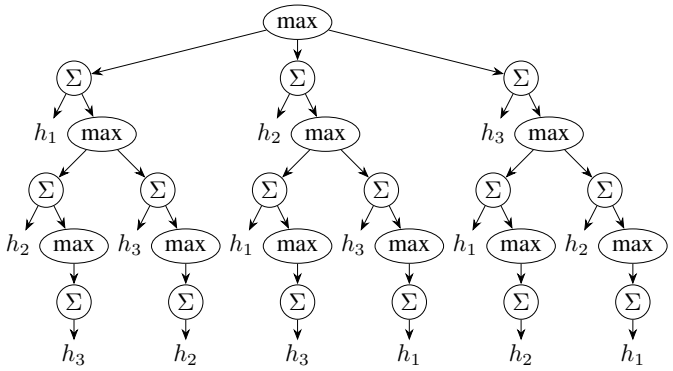
Since ADHGs were introduced for arbitrary collections of heuristics, we must use the correct remaining costs when saturating a heuristic to ensure that the heuristic encoded by the ADHG is admissible. For heuristic h_i, the correct



(a) Naive all-order ADHG for $\mathcal{H}$.



(c) Reduced representation of Figure 1b.



(b) Prefix-merged ADHG for $\mathcal{H}$ if all lookup tables are unique.

Figure 1: Three ADHGs representing the same h^{SCP}_* heuristic for $\mathcal{H} = \{h_1, h_2, h_3\}$ with different levels of compactness. Cost functions are omitted for conciseness, so heuristic h_i at different locations represents the same abstraction heuristics but evaluated under different cost functions.

remaining cost function can be computed as the original cost function minus the saturated costs of any heuristic that appears in a sum node on the path from the root to h_i. This ensures that ADHGs compute an *admissible* heuristic by being *safely additive* (see Definition 4.4 and Theorem 2 by Coles et al. (2008)).

Example 1. *Given the ADHG for heuristics $\{h_1, h_2, h_3\}$ in Figure 1a, the heuristic value of the root node can be calculated by evaluating all saturated cost partitions for the leaf nodes, i.e., for the first sum node we obtain $sum_1(s) = h_1(mscf_1, s) + h_2(mscf_2, s) + h_3(mscf_3, s)$ for all $s \in S$, where $mscf_1 = saturate(h_1, cost)$, $mscf_2 = saturate(h_2, cost - mscf_1)$ and $mscf_3 = saturate(h_3, cost - mscf_1 - mscf_2)$. The overall ADHG heuristic value is then $h^{\text{ADHG}}(s) = \max(sum_1(s) + \cdots + sum_6(s))$.*

The ADHG structure was introduced as a general tool for representing additivity relations between arbitrary heuristics. However, it is particularly well suited for our use case because the common prefix of multiple orders can be captured as common ancestors in a graph. By putting evaluation nodes for shared heuristics higher in the graph, the ADHG can represent common prefixes of multiple orders compactly by creating a hierarchy of evaluation nodes. Figures 1a and 1b show the reduction in representation size when switching to this prefix-merged representation. Conceptually, max nodes represent branching points where a common order prefix splits into different saturation orders. Sum nodes sequentially add more heuristics to the saturation order along a specific branch.

In a fully expanded prefix-merged ADHG (e.g., Figure 1b), the max node in the first layer has n branches and n lookup tables, each node in the second max node layer has $n-1$ branches and $n \cdot (n-1)$ lookup tables, until the nodes in the final layer have no further branches and $n!$ lookup tables. Using the permutation function $P(x, y) = \frac{x!}{(x-y)!}$, the number of lookup tables in layer l is $P(n, l)$. An ADHG therefore reduces the number of lookup tables from the naive $n \cdot n!$ to $\sum_{l=1}^{n} P(n, l) = \sum_{k=0}^{n-1} \frac{n!}{k!}$. Since $\sum_{k=0}^{n-1} \frac{1}{k!}$ approximates Euler's number e, this yields approximately $e \cdot n!$ lookup tables.

3.1 General ADHG Reduction Rules

Previous work used ADHGs to represent small, hand-crafted heuristics (Coles et al. 2008). Our work is, to our knowledge, the first to use ADHGs to represent large, algorithmically generated heuristics with up to millions of nodes. Generating large ADHGs algorithmically makes it important to keep their size as small as possible to allow for efficient evaluation. Therefore, we ensure that an ADHG never includes duplicate lookup tables or structurally equivalent subgraphs. We achieve this by indexing all leaf lookup tables based on their minimum saturated cost function and corresponding abstraction. Then, we index internal nodes based on their type (sum/max) and the index of all their child nodes. While this eliminates structural redundancy, it does not address the remaining mathematical redundancy. To tackle this, we introduce the first general reduction rules for ADHGs that can be used to automatically remove redundant computations. They are general in the sense that they apply to ADHGs over arbitrary types of heuristics, not just those for computing h^{SCP}_*. Every reduction rule can be derived from the properties of the mathematical expression represented by the ADHG. The first proposition follows from the associativity of the sum and max operations.

Proposition 1. *If an ADHG node c has a parent p that is of the same type, then p and c can be combined without changing the ADHG heuristic. This is achieved by adding all children of c as children of p and removing the connection between p and c.*

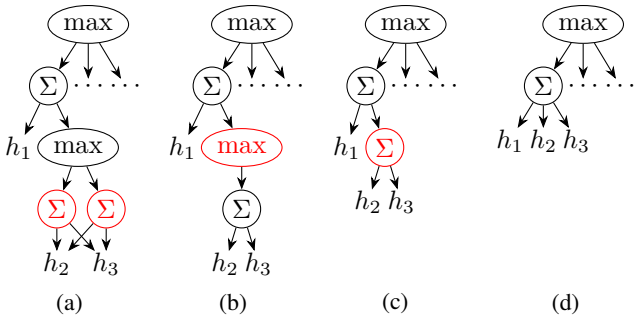


Figure 2: Example showing general reduction rules applied to the ADHG from Figure 1c. Figure 2a shows part of Figure 1c with the change that h_2 and h_3 are order-independent after saturating h_1. Figure 2b shows the ADHG from Figure 2a without duplicated nodes. Applying Proposition 2 to Figure 2b yields the ADHG for Figure 2c and applying Proposition 1 to Figure 2c results in Figure 2d.

An example is shown in the change from Figure 2c to Figure 2d. Consequently, in an ADHG that is fully reduced with respect to Proposition 1, all children of sum nodes are max nodes and the other way around. Furthermore, maximum and sum operations on a single operand are the identity function, which is another point of possible reductions.

Proposition 2. *If an ADHG node n has a single child node c and parent nodes $p_1, \ldots, p_k$, then c can be added as a child to each $p_1, \ldots, p_k$. If n has no parent nodes (i.e., it is the root node), then c becomes the new root node. In both cases, n can be removed from the graph by removing the connections between $p_1, \ldots, p_k$, and n, and the heuristic is unaffected.*

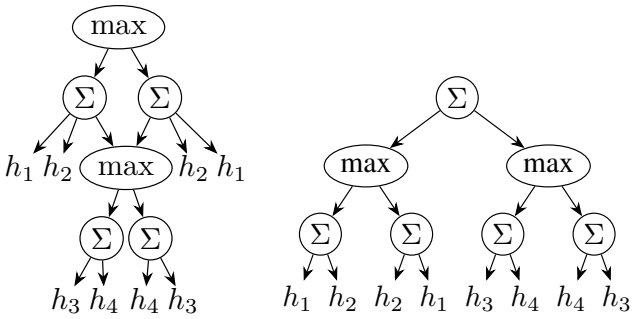
Examples are shown in the change between Figures 1b and 1c and between Figures 2b and Figure 2c. The final general reduction rule in this section simplifies common factors in max nodes and is essential for the proof of Theorem 3.

Proposition 3. *Let* G *be an ADHG that is fully reduced with respect to Proposition 1. If there exists a max or evaluator node node_n in* G *that appears in all sum node children of a max node max_m. Then node_n can be removed from all sum node children of max_m and a new sum node with both max_m and node_n can be inserted between max_n and all its parents.*

Proof. Addition is distributive over the maximum operation. Therefore any term $\max(a + b_1, a + b_2, \ldots, a + b_n)$, where $a, b_1, \ldots, b_n \in \mathbb{R} \cup \{-\infty, \infty\}$, is equal to $a + \max(b_1, \ldots, b_n)$. □

An example for this reduction rule is shown in Figure 3.

Viewing ADHGs as *symbolic expression graphs* with the heuristics as symbolic variables shows that a lot more reduction rules would be possible from exploiting more involved symmetric, associative, and distributive structures. A general solution would be to employ a *term rewriting system* or, more specifically, a *graph rewriting system* (Ehrig et al. 2006) that is capable of performing algebraic simplification on ADHGs. Due to the complexity of these reductions, we leave better optimizations and reductions of ADHGs as future work.



(a) Prefix-merged ADHG only containing orders where $\mathcal{H}_1$ precedes $\mathcal{H}_2$.

(b) ADHG from Figure 3a after applying Proposition 3.

Figure 3: Example of order-independent sets of heuristics leading to less complex ADHGs by applying Theorem 3. The set of heuristics is $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$ and $\mathcal{H}_1 = \{h_1, h_2\}$ is order-independent from $\mathcal{H}_2 = \{h_3, h_4\}$. Note that, although the number of nodes increases from 3a to 3b, the number of mathematical operations required to evaluate the graph decreases.

4 Order Independence Between Heuristics

To further reduce the size of ADHGs, we now exploit the specific properties of ADHGs for SCP. The previous reduction rules were post-hoc reduction rules that simplified the ADHGs nodes after they were generated. Now we introduce a preemptive reduction rule that allows us to skip redundant computations before generating their nodes. It is based on the insight that two different orders ω and ω' for SCP can induce the same SCP heuristic, i.e., $h^{\text{SCP}}_{\omega}(s) = h^{\text{SCP}}_{\omega'}(s)$ for all $s \in S$. Detecting and even predicting this equivalence between SCP heuristics is essential for efficiently computing h^{SCP}_*, as it allows us to drastically reduce the number of orders we need to consider explicitly. Figure 2 shows a graphical example of this: knowing that the order of h_2 and h_3 does not matter after saturating h_1 helps to reduce the size of the ADHG.

Checking whether two orders yield the same SCP heuristic is challenging but can be approximated by comparing the underlying cost partitions.

Proposition 4 (Equivalent Orders)**.** *Given two orders ω and ω' over the same set of heuristics $\mathcal{H}$ and a cost function cost, then $h^{\text{SCP}}_{\omega} = h^{\text{SCP}}_{\omega'}$ if their cost partitions $\mathcal{C}^{\text{SCP}}_{\omega}$ and $\mathcal{C}^{\text{SCP}}_{\omega'}$ assign the same cost function to the same heuristics, e.g., $\mathcal{C}^{\text{SCP}}_{\omega}(h_i) = \mathcal{C}^{\text{SCP}}_{\omega'}(h_i)$ for all $1 \leq i \leq n$. We say ω and ω' are* equivalent *under cost if this holds.*

The proof follows directly from the definition of saturated cost partitioning. Proposition 4 is only a sufficient condition for the equivalence of two SCP heuristics since different cost functions can still result in equivalent SCP heuristics.[1] To relate equivalent orders to abstraction heuristics, we define order-independent heuristics.

Definition 2 (Order Independence)**.** *Two heuristics h_1 and*

[1]For example, if the set of heuristics consists of two instances of the same abstraction heuristic that consumes all costs when it is saturated, both orders will result in the same SCP heuristic, but the two underlying cost partitions are different.

h_2 *are* order-independent *under cost function cost, if* $\omega = \langle h_1, h_2 \rangle$ *and* $\omega' = \langle h_2, h_1 \rangle$ *are equivalent under cost.*

To extend pairwise order independence to sets of heuristics, we introduce the notion of *order-independent sets*.

Definition 3 (Order-Independent Sets)**.** *Let* $\mathcal{H}_I = \{\mathcal{H}_1, \ldots, \mathcal{H}_n\}$ *be a partition of* $\mathcal{H}$*.* $\mathcal{H}_I$ *is an* order-independent set *if and only if all orders of* $\mathcal{H}$*, where the relative order of heuristics within each set* $\mathcal{H}_i$ *is the same, are equivalent under cost. We call heuristic sets that fulfill these conditions* order-independent*.*

Example 2. *Consider the sets* $\mathcal{H} = \{h_1, h_2, h_3\}$*,* $\mathcal{H}_1 = \{h_1, h_2\}$ *and* $\mathcal{H}_2 = \{h_3\}$*. Then* $\{\mathcal{H}_1, \mathcal{H}_2\}$ *is an order-independent set under cost if* $\mathcal{C}^{\text{SCP}}_{\langle h_1, h_2, h_3 \rangle}(h_i) = \mathcal{C}^{\text{SCP}}_{\langle h_1, h_3, h_2 \rangle}(h_i) = \mathcal{C}^{\text{SCP}}_{\langle h_3, h_1, h_2 \rangle}(h_i)$ *and* $\mathcal{C}^{\text{SCP}}_{\langle h_2, h_1, h_3 \rangle}(h_i) = \mathcal{C}^{\text{SCP}}_{\langle h_2, h_3, h_1 \rangle}(h_i) = \mathcal{C}^{\text{SCP}}_{\langle h_3, h_2, h_1 \rangle}(h_i)$ *for all* $1 \leq i \leq n$*.*

Order-independent sets are a special case of order-independent pairs because they require pairwise order independence between all elements in the different order-independent sets.

Theorem 1. *A partition* $\{\mathcal{H}_1, \ldots, \mathcal{H}_n\}$ *of the heuristics* $\mathcal{H}$ *is an order-independent set under cost function cost if every heuristic* $h_i \in \mathcal{H}$*, with* $h_i \in \mathcal{H}_a$*, is order independent of every* $h_j \in \mathcal{H} \setminus \mathcal{H}_a$ *under any cost function that can be obtained by saturating abstractions from* $\mathcal{H} \setminus \{h_i, h_j\}$*.*

Proof. Given a partition of $\mathcal{H}$, every order that preserves the relative positioning of heuristics inside each partition can be reached by swapping two order-adjacent heuristics $h_1 \in \mathcal{H}_a$ and $h_2 \in \mathcal{H}_b$. Each such swap preserves the overall heuristic value if h_1 and h_2 are order-independent under the remaining costs after saturating all $h_i \in \mathcal{H} \setminus \{h_1, h_2\}$ that appear before h_1 and h_2 in the order. □

Theorem 1 shows that it is sufficient to check order independence between pairs of heuristics to detect order-independent sets if the order-independence holds for all cost functions that can occur during the saturation process. To compute order-independent sets of heuristics from pairwise order-independence information, we build a graph that reflects the opposite information, i.e., order *dependence*.

Definition 4 (Order-Dependence Graph)**.** *The order-dependence graph* $G_D(\mathcal{H}, cost) = \langle V, E \rangle$ *for saturated cost partitioning over heuristics* $\mathcal{H}$ *and cost function cost has a vertex* $h \in V$ *for each heuristic* $h \in \mathcal{H}$ *and an edge* $\langle h_1, h_2 \rangle \in E$ *if* $h_1 \in \mathcal{H}$ *and* $h_2 \in \mathcal{H}$ *are order-dependent under any remaining cost function that can be reached by saturating abstractions from* $\mathcal{H} \setminus \{h_1, h_2\}$*.*

Given an order-dependence graph G_D, we can compute its connected components $\mathcal{H}_1, \ldots, \mathcal{H}_n$ which form order-independent heuristic sets.

Theorem 2. *The connected components* $\mathcal{H}_1, \ldots, \mathcal{H}_n$ *of an order-dependence graph* $G_D(\mathcal{H}, cost)$ *are order-independent heuristic sets under cost.*

Proof. Consider two connected components, $\mathcal{H}_1$ and $\mathcal{H}_2$, of an order-dependence graph $G_D(\mathcal{H}, cost)$. The heuristic sets are *not* order-independent if (1) $\mathcal{H}_1$ and $\mathcal{H}_2$ are not disjoint, or (2) there exists a heuristic $h \in \mathcal{H}$ and a heuristic $h' \in \mathcal{H}'$ that are order-dependent under *cost*. However, by definition, two connected components of a graph are disjoint (contradicting Condition 1). Additionally, according to Definition 4, if two heuristics h and h' are order-dependent under *cost*, there is an edge $\langle h, h' \rangle$ in $G_D(\mathcal{H}, cost)$, meaning h and h' cannot belong to different components (contradicting Condition 2). Thus, any two heuristics of two separate components are order-independent, which by Theorem 1 shows that the connected components of the order-dependence graph form order-independent heuristic sets. □

As computing connected components of a graph requires only quadratic time in the graph size, this is a practical approach to determining order-independent heuristic sets.

5 An Order Independence Reduction Rule

Above, we theoretically defined order-independent sets and showed how to construct them from order-independent pairs of heuristics. Now, we derive a reduction rule specific to h^{SCP}_* for ADHGs exploiting order-independent sets. This rule allows us to avoid constructing the full ADHG for a set of order-independent heuristics by directly constructing an equivalent, more concise ADHG.

To reduce the size of an ADHG for h^{SCP}_*, we aim to exclude all redundant orders. For equivalent orders, this can be achieved by including only one order over the order-independent sets. We choose an arbitrary order for the independent sets, $\mathcal{H}_1, \ldots, \mathcal{H}_n$ and use the order where all elements of $\mathcal{H}_i$ are placed before all elements of $\mathcal{H}_{i+1}$. There always exists one order in each class of equivalent orders that fulfills this requirement, as the definition of an order-independent set is that the order between independent sets is swappable. Including only one order for each independent set in an ADHG creates a new reduction opportunity. As the order of putting $\mathcal{H}_i$ before $\mathcal{H}_{i+1}$ is artificial and does not affect the heuristic value, we can completely remove the ordering between different order independent sets. This is achieved by summing over each order-independent set. We now show that this reformulation doesn't change the result of the ADHG and is equivalent to applying Proposition 3 on the original ADHG.

Theorem 3. *Given order-independent sets* $\mathcal{H}_1, \ldots, \mathcal{H}_n$*, let* G *be the prefix-merged ADHG that includes only the orders where all elements of* $\mathcal{H}_i$ *are placed before all elements of* $\mathcal{H}_{i+1}$*. Then Proposition 3 can be applied* $n - 1$ *times until* G *has a sum node over one subgraph for each* $\mathcal{H}_i$*.*

Proof. G contains equivalent subgraphs for all orders of $\mathcal{H}_{i+1}$ under every subgraph for $\mathcal{H}_i$, as regardless of the order chosen for $\mathcal{H}_i$, the abstraction heuristics from $\mathcal{H}_{i+1}$ will receive the same costs as per Definition 3. This means that the max node $max_{\mathcal{H}_i}$ that branches over all orders of $\mathcal{H}_i$ contains the same subgraph $max_{\mathcal{H}_{i+1}}$ in each of its branches and Proposition 3 can be applied. This can be repeated for all $\mathcal{H}_i$ where $i > 1$. The resulting ADHG has a sum node over one subgraph for each $\mathcal{H}_i$. □

Given a set of order-independent heuristics, we can therefore directly generate the reduced graph instead of first generating the semi-reduced graph and then iteratively applying Proposition 3. The following is an example of the reduction described in Theorem 3.

Example 3. *Consider the set of heuristics $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$, where $\mathcal{H}_1 = \{h_1, h_2\}$ is order-independent from $\mathcal{H}_2 = \{h_3, h_4\}$ under cost. Figure 3 shows two ADHGs representing h^{SCP}_* for $\mathcal{H}$. The first ADHG is a prefix-merged representation that considers all heuristics from $\mathcal{H}_1$ before those from $\mathcal{H}_2$. The second ADHG applies Proposition 3 as follows: in Figure 3a, the term $c := \max(h_3 + h_4, h_4 + h_3)$ is a common summand in both sum subgraphs. Therefore, the formula $\max(h_1 + h_2 + c, h_2 + h_1 + c)$ can be simplified to $\max(h_1 + h_2, h_2 + h_1) + c = \max(h_1 + h_2, h_2 + h_1) + \max(h_3 + h_4, h_4 + h_3)$. Remember that summation is not commutative here, because the order of the heuristics influences the resulting cost partitioning and therefore the heuristic value.*

By exploiting order-independence in this way, we reduce the ADHG size from $O(|\mathcal{H}|!)$ to a possibly much smaller $O(|\mathcal{H}_1|! + \cdots + |\mathcal{H}_n|!) = O(\max_i^n |\mathcal{H}_i|!)$. Of course, we can also exploit order independence between sets of heuristics in subgraphs of ADHGs where the positions of some heuristics are already decided, as shown in Figure 2.

6 Computing Order Independence

Having discussed how to use order-independence sets for reducing the ADHG size, we now define several sufficient criteria guaranteeing that two abstraction heuristics are order-independent. As mentioned in Section 4, we are interested in cases where a change in saturation order does not affect the cost partitions. To capture order independence, we therefore need to define conditions under which an abstraction heuristic yields the same cost partitioning for different cost functions.

Theorem 4. *Two abstraction heuristics h_1 and h_2 are order-independent under cost function cost, if their minimum saturated cost functions are non-negative and form a cost partition.*

Proof. Let h_1 and h_2 be two abstraction heuristics, and let $\mathit{mscf}_1 = \mathit{saturate}(h_1, \mathit{cost})$ and $\mathit{mscf}_2 = \mathit{saturate}(h_2, \mathit{cost})$ be their minimum saturated cost functions. For order-independence it needs to hold that the two orders $\omega = \langle h_1, h_2 \rangle$ and $\omega' = \langle h_2, h_1 \rangle$ are equivalent, i.e., the induced cost partitions $\mathcal{C}^{\text{SCP}}_{\omega}$,$\mathcal{C}^{\text{SCP}}_{\omega'}$ are equivalent. This is the case if $\mathit{saturate}(h_1, \mathit{cost}) = \mathit{saturate}(h_1, \mathit{cost} - \mathit{mscf}_2)$ and $\mathit{saturate}(h_2, \mathit{cost}) = \mathit{saturate}(h_2, \mathit{cost} - \mathit{mscf}_1)$. Assume that mscf_1 and mscf_2 are non-negative, i.e., no label is assigned negative costs, and that they form a cost partition under *cost*, i.e., $\mathit{mscf}_1 + \mathit{mscf}_2 \leq \mathit{cost}$. Given the latter, saturating h_2 under the remaining cost $\mathit{rem} = \mathit{cost} - \mathit{mscf}_1$ offers each label at least the cost that h_2 needs for its minimum saturated cost function:

$$\begin{aligned} \mathit{cost} \geq \mathit{mscf}_1 + \mathit{mscf}_2 &\Leftrightarrow \mathit{cost} - \mathit{mscf}_1 \geq \mathit{mscf}_2 \\ &\Leftrightarrow \mathit{rem} \geq \mathit{mscf}_2 \end{aligned} \tag{1}$$

When assuming *non-negativity* of the minimum saturated cost functions, then saturating h_2 under *rem* offers each label at most the cost that h_2 is offered by the original cost function.

$$\mathit{mscf}_1 \geq 0 \Leftrightarrow \mathit{cost} - \mathit{mscf}_1 \leq \mathit{cost} \Leftrightarrow \mathit{rem} \leq \mathit{cost} \tag{2}$$

In combination, Equations 1 and 2 bound the costs after saturating h_1 from above and below: $\mathit{mscf}_2 \leq \mathit{rem} \leq \mathit{cost}$. Since changing the cost function of an abstraction to its saturated cost function does not change the abstract goal distance of any state, it holds that $h_2(\mathit{cost}, s) = h_2(\mathit{mscf}_2, s) = h_2(\mathit{rem}, s)$ for all states $s \in S$, from which it follows that $\mathit{saturate}(h_2, \mathit{cost}) = \mathit{saturate}(h_2, \mathit{rem})$. The argument for h_1 follows from symmetry. □

The order independence conditions in Theorem 4 have two disadvantages. First, they are based on the minimum saturated cost function of a heuristic. Computing the minimum saturated cost function necessitates calculating the lookup tables, i.e., abstract goal distances, of an abstraction. Therefore, it cannot alleviate the costly computation of the lookup tables, making it impractical for determining order independence. Second, they depend on the original cost function. This makes it difficult to derive the order-independence of two heuristics $h_1, h_2 \in \mathcal{H}$ because we need to compute the remaining cost functions that can be obtained by saturating the heuristics $\mathcal{H} \setminus \{h_1, h_2\}$ (Theorem 1).

In the following, we describe two sufficient conditions derived from Theorem 4 that do not rely on knowing the exact cost function. The first condition exploits that infinite label costs can never cause order dependence.

Theorem 5 (SCP*-∞)**.** *If the cost of an operator ℓ is infinite in cost function cost, then ℓ is never the cause of order-dependence under any cost function that can be reached by subtracting saturated costs from cost.*

Proof. If $\mathit{cost}(\ell) = \infty$, saturating any heuristic will always leave the remaining costs for ℓ infinite, since cost functions use left addition. Therefore, all abstractions will always receive the same cost for ℓ regardless of saturation order. □

For the second condition, we exploit the observation that two heuristics always form a cost partition if for every label at least one of the minimum saturated cost functions is guaranteed to be zero under any cost function.

Definition 5 (Saturation-Affecting Labels)**.** *A label ℓ is* saturation-affecting *for an abstraction heuristic h if there exists a cost function cost under which the minimum saturated cost $\mathit{saturate}(h, \mathit{cost})(\ell)$ is neither zero nor negative infinity.*[2] *We write $L_h^{\mathit{sat\text{-}aff}}$ for the set of saturation-affecting labels for heuristic h.*

It is easy to verify that every saturation-affecting label is also affecting. To see that the opposite is not true, consider a label ℓ that only labels transitions between goal states. Then ℓ is affecting, but not saturation-affecting. As we are interested

[2]This formulation is similar to the affecting labels of Seipp, Keller, and Helmert (2017a) when extended to general cost functions (allowing negative costs), and explicitly states the connection to minimum saturated cost functions.

in labels that are not saturation-affecting for any abstraction heuristic, it is sufficient to show that the sets of saturation-affecting labels do not overlap.

Lemma 1. *If two heuristics h_1 and h_2 have disjoint sets of saturation-affecting labels, they are order-independent under any cost function.*

Proof. If two heuristics h_1 and h_2 have disjoint sets of saturation-affecting labels, none of the label costs that change upon saturating h_1 influence the heuristic values of h_2, and vice versa. Formally, for every label ℓ either $mscf_1(\ell)$ or $mscf_2(\ell)$ is zero or infinity, so the cost partitioning condition is fulfilled. The minimum saturated cost functions do not need to be *non-negative*, since having more cost available does not change the minimum saturated costs for a label that is not saturation-affecting. □

To approximate if two heuristics have disjoint sets of saturation-affecting labels, we overapproximate the set of saturation-affecting labels for a heuristic.

Theorem 6 (SCP*-AFF)**.** *Let $L_h^{\text{non-goal}}$ be the set of labels $\ell \in L$ for which there is at least one transition $s \xrightarrow{\ell} s' \in T^\alpha$ between two distinct states s and s', of which at most one state is a goal state. If the sets $L_{h_1}^{\text{non-goal}}$ and $L_{h_2}^{\text{non-goal}}$ for two heuristics h_1 and h_2 are disjoint, then h_1 and h_2 are order-independent under any cost function cost.*

Proof. If label ℓ affects a heuristic h^α, then by the definition of minimum saturated cost functions, there must be two abstract states s and s', such that $h^*_{\mathcal{T}^\alpha}(s) \ominus h^*_{\mathcal{T}^\alpha}(s') \neq 0$. So $h^*_{\mathcal{T}^\alpha}(s) \neq h^*_{\mathcal{T}^\alpha}(s')$ or $h^*_{\mathcal{T}^\alpha}(s) = h^*_{\mathcal{T}^\alpha}(s') = \infty$. (Abstract goal distances are always non-negative, so they cannot be $-\infty$.) In both cases, both states cannot be goal states at the same time, so $L^{\text{non-goal}} \subseteq L^{\text{sat-aff}}$. Therefore, if $L_{h_1}^{\text{non-goal}} \cap L_{h_2}^{\text{non-goal}} = \emptyset$ then $L_{h_1}^{\text{sat-aff}} \cap L_{h_2}^{\text{sat-aff}} = \emptyset$. Using Lemma 1 it follows that h_1 and h_2 are order-independent. □

Combining the conditions of Theorems 5 and 6 yields an efficient approximation of the order independence of two heuristics.

Theorem 7 (SCP*-AFF-∞)**.** *For heuristics h_1 and h_2, let $L_{h_1}^{\text{non-goal}}$ and $L_{h_2}^{\text{non-goal}}$ be sets of labels as in Theorem 6. Then h_1 and h_2 are order-independent, if*

$$(L_{h_1}^{\text{non-goal}} \cap L_{h_2}^{\text{non-goal}}) \setminus \{\ell \in L \mid \mathit{cost}(\ell) = \infty\} = \emptyset.$$

All of the order independence results from Theorem 5 to 7 can, in contrast to Theorem 4, be used for pruning connections in the order-dependence graph, as they generalize to all cost functions that can be reached by subtracting saturated costs from the original cost function. When creating an ADHG with any of the introduced approximation methods, we repeatedly test the order independence of the remaining, not yet saturated heuristics at each node.

Example 4. *Consider a set of heuristics, $\mathcal{H} = \{h_1, h_2, h_3\}$, where the used order-independence approximation detects no order independence. Instead of iterating over all $|\mathcal{H}|! = 6$ orders, we create a single max node with one child sum node c_i per heuristic h_i. Each node c_i has as first child the lookup table for h_i under the original cost function. The other children of c_i will become the max nodes m_i for the sets of remaining heuristics $\mathcal{H} \setminus \{h_i\}$. For example, for c_1 the remaining heuristics are h_2 and h_3. In this situation, h_2 and h_3 may have become order-independent for two reasons. First, because h_1 was removed from the order dependence graph, h_2 and h_3 might have become disconnected. Second, saturating for h_1 changed the remaining cost function at max node m_1. This means that testing the order-independence condition again might now detect h_2 and h_3 as order-independent, even though they were not order-independent before.*

Recomputing the order-dependence graph is cheap because Theorem 6 is independent of any cost function. Therefore, the order-dependence graph does not need to be re-computed when the remaining costs or the set of heuristics change. Since Theorem 5 is a simple check, the combination (Theorem 7) is also cheaply computable.

Further Order Independence Conditions We described two sufficient conditions for checking order-independence of heuristics. Other conditions for order-independence can be derived by approximating the conditions in Theorem 4 in different ways. While such investigations are interesting, we leave them for future work.

7 Experiments

The ADHG data structure and our reduction rules enable us to compute h_*^{SCP} without explicitly considering all possible orders of the component heuristics. The main questions we aim to answer with our experiments are:

- How helpful is the ADHG structure compared to a naive computation of h_*^{SCP}?
- How helpful are the order-independence approximations?
- How good are existing approximations of h_*^{SCP}?
- How close is h_*^{SCP} to optimal cost partitioning?

7.1 Experimental Setup

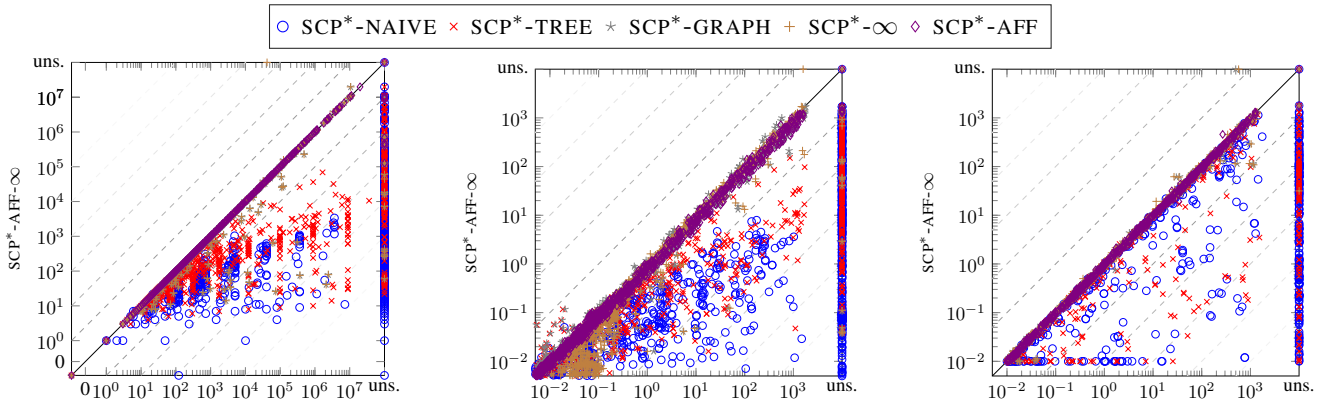
We implemented the presented approaches in the Scorpion planner (Seipp, Keller, and Helmert 2020), an extension of Fast Downward (Helmert 2006). For all experiments, we use Cartesian goal abstractions (Seipp and Helmert 2018), which are known to provide a good trade-off between the number of abstractions and the informativeness of the resulting h^{SCP} heuristic. Our benchmark set includes all planning tasks without conditional effects and axioms from the Optimal Tracks of the International Planning Competitions (IPCs) 1998–2023. All runs have time and memory limits of 30 minutes and 8 GiB, respectively. We run experiments using Downward Lab (Seipp et al. 2017) and make all code and experimental data available online (Höft, Speck, and Seipp 2025).

7.2 Computing the Perfect SCP Heuristic

In the first experiment, we compare different ways of computing h_*^{SCP}. The configurations vary in their treatment of

		Other Heuristics			Variants for Computing h^{SCP}_*					
		h^{OCP}	$h^{\text{SCP}}_{\text{div}}$	$h^{\text{SCP}}_{\text{rnd}}$	SCP*-NAIVE	SCP*-TREE	SCP*-GRAPH	SCP*-∞	SCP*-AFF	SCP*-AFF-∞
initialize	finished	1685	1884	1884.0	1101	1313	1550	1551	1560	1568
	out of time	192	0	0.0	47	309	254	283	271	261
	out of memory	7	0	0.0	736	262	80	50	53	55
search	solved	499	1080	993.4	728	872	983	982	982	983
	out of time	1186	40	6.2	173	149	160	165	157	156
	out of memory	0	764	884.4	200	292	407	404	421	429

Table 1: Summary of outcomes for optimal cost partitioning, two approximative SCP algorithms, and the h^{SCP}_* algorithms. All configurations use Cartesian goal abstractions for a benchmark suite of 1884 planning tasks. The initialization part consists of all operations needed to compute $h(s_0)$. We average results for $h^{\text{SCP}}_{\text{rnd}}$ over five random seeds.



(a) Representation size of the final heuristic (ADHG nodes).

(b) Time to initialize the heuristic and compute $h(s_0)$.

(c) Search time for solving the planning tasks (after initialization).

Figure 4: Comparison of h^{SCP}_* variants. Each point represents a comparison of SCP*-AFF-∞ on the y-axis with another variant indicated by color on the x-axis, for a particular planning task. Points below the diagonal indicate that SCP*-AFF-∞ is preferable for that task (4a: smaller representation size, 4b: faster heuristic initialization, 4c: faster problem solving).

order independence among heuristics and node sharing in the ADHG:

1) SCP*-NAIVE enumerates all heuristic orders.
2) SCP*-TREE represents h^{SCP}_* as a tree, ignores order independence, and applies only the general reduction rules from Section 3.1.
3) SCP*-GRAPH is like SCP*-TREE, but shares equivalent internal nodes, resulting in a graph.
4) SCP*-∞ uses a graph and exploits order independence between heuristics, applying only the infinite cost rule from Theorem 5.
5) SCP*-AFF uses a graph and exploits order independence between heuristics, applying only the saturation-affecting labels rule from Theorem 6.
6) SCP*-AFF-∞ combines the latter two approaches. It uses a graph and applies both order-independence rules of SCP*-∞ and SCP*-AFF, i.e., the rule from Theorem 7.

The *initialize finished* row in Table 1 shows how often it is possible to construct the data structures required to compute the initial heuristic value $h^{\text{SCP}}_*(s_0)$ using the naive and ADHG-based approaches, as configurations become more sophisticated. The naive approach SCP*-NAIVE which considers all orders explicitly is infeasible for larger tasks, because it usually runs out of memory during initialization as soon as more than ten abstractions are present ($\geq 3\,628\,800$ orders).

The table also shows that building h^{SCP}_* as a graph instead of a tree significantly raises the number of finished initializations, and adding the independence conditions increases this number even more. Although our more advanced configurations run out of time more often than the naive variant, they do not incur the same memory bottleneck for constructing the data structure of the heuristic. Remarkably, we can compute $h^{\text{SCP}}_*(s_0)$ with SCP*-AFF-∞ even for tasks with up to 160 abstractions and thus $\sim 10^{284}$ orders. However, due to the factorial nature of orders, computing h^{SCP}_* within the resource limits remains very challenging.

Figures 4a, 4b, and 4c visualize the representation size of the heuristic (ADHG nodes), the time it takes to create the heuristic representation, and the resulting A* search time for the different approaches. The plots reveal a significant advantage of the graph representation over the tree and naive variants across the benchmark tasks. Using a graph instead of a tree leads to the reduction of the number of nodes in the

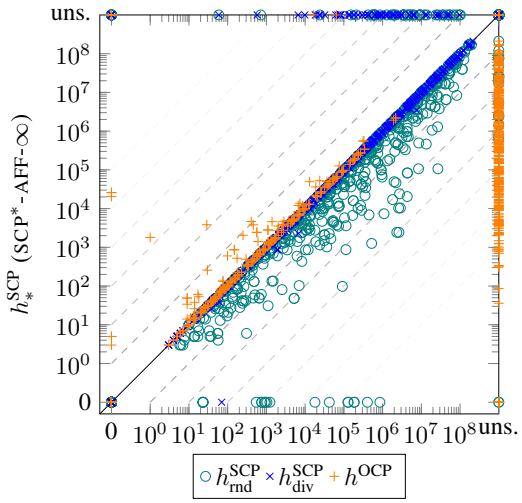


Figure 5: Necessary state expansions (i.e., expansions before the last f-layer) of the perfect SCP heuristic, those of approximative SCP heuristics and the optimal cost partitioning heuristic.

ADHG by up to four orders of magnitude (Figure 4a). This trend is also visible in the time it takes to create the necessary data structures in Figure 4b.

Considering search performance, the number of solved tasks is the same for both SCP*-GRAPH and SCP*-AFF-∞, and decreases by one if only one of the order-independent rules is applied (SCP*-∞ and SCP*-AFF). However, the advantage of using order independence conditions is evident in the ability to generate the ADHGs on larger abstraction sets. This is shown in the higher number of finished initializations in Table 1 and the large number of unsolved tasks on the x-axis in Figure 4a. This shows that, while those rules help to compute h_*^{SCP} for more tasks, the underlying tasks are too challenging to solve. Finally, Figure 4c compares the time it takes to solve planning tasks, showing that the tree and naive approaches are typically slower than SCP*-AFF-∞ due to their larger representation sizes. Comparing SCP*-AFF-∞ and SCP*-GRAPH, the order-independent rules have no significant positive or negative effect on runtime, similar to the representation sizes.

7.3 Heuristic Accuracy

Now that we can efficiently compute h_*^{SCP}, we can, for the first time, evaluate the approximation quality of approximative SCP heuristics from the literature. For this analysis, we use our best approach SCP*-AFF-∞ to compute h_*^{SCP}, which takes advantage of all presented optimizations.

In particular, we compare against the approximative SCP heuristics $h_{\text{rnd}}^{\text{SCP}}$ and $h_{\text{div}}^{\text{SCP}}$. The former considers a single random ordering of the abstractions (reported results are averages over five runs). The latter iteratively samples a new state s until hitting a fixed time limit of T=100 seconds, approximates a useful order ω for s with a greedy algorithm, and retains h_ω^{SCP} if it is *diverse*, that is, it yields a higher estimate for any of a set of 1000 independently sampled states than the SCP heuristics already retained previously (Seipp, Keller, and Helmert 2020). We also empirically compare the perfect SCP heuristic h_*^{SCP} with the optimal cost partitioning heuristic h^{OCP} (Pommerening et al. 2015), which is known to theoretically dominate h_*^{SCP} (Pommerening et al. 2015; Seipp, Keller, and Helmert 2017a).

The left part of Table 1 holds results for these heuristics and Figure 5 shows the necessary node expansions, i.e., the expansions before the last f-layer, for the compared approaches. Comparing h_*^{SCP} with $h_{\text{rnd}}^{\text{SCP}}$, we see that considering all orders instead of a single random one yields significantly more informative heuristic estimates, reducing the necessary expansions by a large margin. However, the picture is different for the state-of-the-art approach of computing diverse orders, $h_{\text{div}}^{\text{SCP}}$. Although h_*^{SCP} has fewer node expansions than $h_{\text{div}}^{\text{SCP}}$ for 334 tasks, the difference in heuristic accuracy between h_*^{SCP} and $h_{\text{div}}^{\text{SCP}}$ is small. This shows that, considering the IPC domains and Cartesian goal abstractions, 1) only a few orders are necessary to obtain a heuristic estimate close to h_*^{SCP}, 2) $h_{\text{div}}^{\text{SCP}}$ is an extremely efficient approach to obtaining relevant orders, and 3) in practice there is little improvement in considering more orders than $h_{\text{div}}^{\text{SCP}}$ when using a moderate number of abstraction heuristics. Comparing the optimal cost partitioning heuristic h^{OCP} and the perfect SCP heuristic h_*^{SCP}, we see that having better cost partitions can indeed yield stronger heuristic estimates that are practically relevant due to a significant reduction in necessary expansions. However, for most tasks where we can compute h^{OCP} the difference is small.

8 Conclusions and Future Work

In this work, we studied the perfect saturated cost partitioning heuristic, which maximizes over individual SCP heuristics computed for all orders of the component heuristics. We showed that naively computing h_*^{SCP} is infeasible due to the inherent factorial nature of the problem. Based on this, we established a connection to additive-disjunctive heuristic graphs and showed that they are well suited to address the computational challenge of computing h_*^{SCP}. By computing order-independent information to exclude provably equivalent orders, we were able to significantly reduce the size of the ADHGs. This representation allowed us to compute h_*^{SCP} for the first time for many planning tasks with a meaningful number of component heuristics. Building on this, we conducted the first empirical comparison of the perfect SCP heuristic, h_*^{SCP}, with state-of-the-art sampling-based SCP heuristics and the optimal cost partitioning heuristic. Our experiments show that state-of-the-art SCP heuristics provide heuristic information that is nearly perfect, relative to their upper bound of h_*^{SCP}, when using a moderate number of abstraction heuristics. This suggests that the greatest potential for obtaining even stronger cost partitioning heuristics lies in developing methods that go beyond SCP and reduce the remaining gap to optimal cost partitioning.

To scale to larger sets of heuristics, we would like to study and derive additional ADHG reduction rules based on the relationship between ADHGs and symbolic expression graphs (Ehrig et al. 2006). Furthermore, finding new order-independence conditions that reduce the size of the ADHG representing h_*^{SCP} is an interesting direction.

Acknowledgments

The authors are grateful to Thomas Keller for the initial idea and implementation for this work and would like to thank Elliot Gestrin and Mika Skjelnes for helpful feedback on the manuscript. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council through grant agreement no. 2022-06725. David Speck was funded by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA).

References

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Additive-disjunctive heuristics for optimal planning. 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)*, 44–51. AAAI Press.

Drexler, D.; Gnad, D.; Höft, P.; Seipp, J.; Speck, D.; and Ståhlberg, S. 2023. Ragnarok. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Ehrig, H.; Ehrig, K.; Prange, U.; and Taentzer, G. 2006. *Fundamentals of Algebraic Graph Transformation*. Springer-Verlag, 1st edition.

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.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

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

Höft, P.; Speck, D.; Pommerening, F.; and Seipp, J. 2024. Versatile cost partitioning with exact sensitivity analysis. In Bernardini, S., and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 276–280. AAAI Press.

Höft, P.; Speck, D.; and Seipp, J. 2023. Sensitivity analysis for saturated post-hoc optimization in classical planning. In Gal, K.; Nowé, A.; Nalepa, G. J.; Fairstein, R.; and Rădulescu, R., eds., *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, 1044–1051. IOS Press.

Höft, P.; Speck, D.; and Seipp, J. 2025. Code and data for the KR 2025 paper "Representing Perfect Saturated Cost Partitioning Heuristics in Classical Planning". https://doi.org/10.5281/zenodo.16606498.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733. AAAI Press.

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

Kreft, R.; Büchner, C.; Sievers, S.; and Helmert, M. 2023. Computing domain abstractions for optimal classical planning with counterexample-guided abstraction refinement. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*, 221–226. AAAI Press.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.

Pommerening, F.; Keller, T.; Halasi, V.; Seipp, J.; Sievers, S.; and Helmert, M. 2021. Dantzig-Wolfe decomposition for cost partitioning. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 271–280. AAAI Press.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364. AAAI Press.

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-guided abstraction refinement for pattern selection in 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)*, 362–367. 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.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Seipp, J.; Keller, T.; and Helmert, M. 2017a. A comparison of cost partitioning algorithms for optimal classical planning. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference*

on Automated Planning and Scheduling (ICAPS 2017), 259–268. AAAI Press.

Seipp, J.; Keller, T.; and Helmert, M. 2017b. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In Singh, S., and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3651–3657. AAAI Press.

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. 2017. Better orders for saturated cost partitioning in optimal classical planning. In Fukunaga, A., and Kishimoto, A., eds., *Proceedings of the 10th Annual Symposium on Combinatorial Search (SoCS 2017)*, 149–153. AAAI Press.

Seipp, J. 2019. Pattern selection for optimal classical planning with saturated cost partitioning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5621–5627. IJCAI.

Seipp, J. 2023. Scorpion 2023. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Sievers, S., and Helmert, M. 2021. Merge-and-shrink: A compositional theory of transformations of factored transition systems. *Journal of Artificial Intelligence Research* 71:781–883.

Sievers, S.; Keller, T.; and Röger, G. 2024. Merging or computing saturated cost partitionings? a merge strategy for the merge-and-shrink framework. In Bernardini, S., and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 541–545. AAAI Press.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Magazine* 45(2):280–296.