# A Comparison of Cost Partitioning Algorithms for Multiple Sequence Alignment

**Mika Skjelnes**[1], **Daniel Gnad**[2, 1], **Jendrik Seipp**[1]

[1]Linköping University, Sweden
[2]Heidelberg University, Germany
firstname.lastname@liu.se, daniel.gnad@uni-heidelberg.de

## Abstract

Cost partitioning is a powerful technique for combining admissible heuristics in optimal classical planning. We study cost partitioning for multiple sequence alignment (MSA), where transition costs decompose into pairwise substitution scores rather than a single scalar action cost. To handle this factored structure, we formulate cost partitioning over *cost components*. While previous work has already adapted three basic cost partitioning algorithms to MSA, our generalized formulation allows us to consider state-of-the-art variants: *saturated cost partitioning* (SCP) and *saturated post-hoc optimization* (SPhO). On the theoretical side, we show that in MSA, SPhO equals PhO, the variant without *cost saturation*, and that for all traditional MSA heuristics we can construct dominating cost partitioning heuristics. On the empirical side, an evaluation on BaliBase benchmarks shows that these dominating cost partitioning algorithms significantly reduce search effort compared to established MSA heuristics.

## Introduction

Multiple sequence alignment (MSA) is a fundamental problem in bioinformatics with applications ranging from studying the evolutionary relationships between biological entities to protein structure prediction and drug design (Carrillo and Lipman 1988; Kobayashi and Imai 1998). Given a set of biological sequences, the goal is to identify regions of similarity that may indicate functional, structural, or evolutionary relationships between the sequences (Carrillo and Lipman 1988). While aligning two or three sequences can be efficiently solved using dynamic programming, the MSA problem becomes computationally challenging as the number of sequences grows (Needleman and Wunsch 1970).

An alignment $A$ of a set of sequences $Q$ adds gaps to the sequences so that they all have the same length. Each row of $A$ corresponds to a modified sequence and each column represents aligned characters, as shown in Figure 2. An optimal alignment minimizes the alignment cost, computed from the sum-of-pairs score (Kobayashi and Imai 1998), which for each column of the alignment sums the substitution costs across all pairs of characters in that column. Exact methods based on A$^*$ search find optimal solutions by using heuristics derived from optimal solutions to subproblems involving small subsets of sequences, such as pairs or triples of sequences (Ikeda and Imai 1994; Kobayashi and Imai 1998).

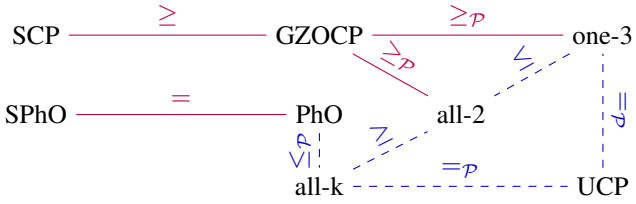



Figure 1: Theoretical dominance relations between heuristics for MSA. $\mathcal{P}$ next to a relation indicates that it holds under a specific pattern collection. Purple solid lines indicate new results presented in this work, blue dashed lines indicate previously known results.

Approaches such as *all-k* combine multiple heuristics (Kobayashi and Imai 1998), but they are inflexible; they aggregate heuristic estimates in predetermined ways, and do not take advantage of the specific structure of the problem instance nor the properties of individual heuristics.

*Cost partitioning*, which was originally developed in the classical planning and heuristic search communities, offers a systematic approach for combining multiple heuristics admissibly (Yang et al. 2008; Katz and Domshlak 2010; Pommerening et al. 2015; Seipp, Keller, and Helmert 2017). The key idea is to distribute the cost of a transition label among multiple heuristics in such a way as to maintain admissibility while maximizing the value of the heuristic at each node, thereby minimizing the number of nodes a search for an optimal solution must expand. Riesterer (2018) and Skjelnes, Gnad, and Seipp (2024) adapted *greedy zero-one cost partitioning* (GZOCP) (Haslum, Bonet, and Geffner 2005; Edelkamp 2006), *post-hoc optimization* (PhO) (Pommerening et al. 2014) and *uniform cost partitioning* (UCP) (Katz and Domshlak 2010) to MSA. However, all three algorithms have counterparts in the classical planning literature that can only yield better heuristic estimates. These algorithms, which are based on *cost saturation*, are critically missing from the MSA literature.

We fill this gap by providing a comprehensive study of cost partitioning for MSA. Our theoretical results are summarized in Figure 1. First, we adapt saturated cost partitioning (SCP) (Seipp, Keller, and Helmert 2020) and saturated post-hoc optimization (SPhO) (Seipp, Keller, and Helmert

| $Q$ | | | | $A$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $s_1$ : | T | T | A | $A_1$ : | T | T | – | – | A |
| $s_2$ : | G | C | | $A_2$ : | – | – | G | – | C |
| $s_3$ : | A | C | | $A_3$ : | A | – | – | C | – |

Figure 2: An alignment $A$ of sequences $Q = \{TTA, GC, AC\}$ (example from Skjelnes, Gnad, and Seipp (2024)).

2021) to MSA. These adaptations are non-trivial because unlike common planning formalisms, MSA exhibits *factored transition costs*: the cost of a transition is determined by the substitution costs that an alignment column incurs. Skjelnes, Gnad, and Seipp (2024) account for this by modeling these factors via *cost components*, each representing the interaction of a specific pair of sequences. We build on this model and generalize the cost partitioning algorithms to operate on these components.

Second, we prove that SPhO is equal to PhO in MSA, even though SPhO is significantly more informative than PhO in classical planning. We also show all MSA-specific heuristics are dominated by at least one cost partitioning heuristic.

Finally, we provide a comprehensive empirical analysis of the discussed algorithms. We see that SCP can improve the heuristic quality over its non-saturating counterpart GZOCP, but this does not make up for the additional computational overhead. Our experiments also show that cost partitioning heuristics in general can significantly improve search performance compared to the traditional MSA-specific heuristics.

## Multiple Sequence Alignment

Given a set of sequences $Q = \{s_1, \dots, s_n\}$ over some alphabet $\Sigma$, an *alignment* of $Q$ is a matrix $A^{n \times m}$ over alphabet $\Sigma' = \Sigma \cup \{-\}$, such that the following conditions hold: (1) $m \geq \max_{s_i \in Q} |s_i|$, (2) no column $j$ consists of only the *gap* character "–" and (3) removing all gap occurrences from any row $i$ in $A$ yields the original sequence $s_i$. The *cost* of an alignment is the summed pairwise substitution cost $sub : \Sigma' \times \Sigma' \to \mathbb{N}_0^+$ of each pair of characters per column in the alignment across all columns. The concrete substitution cost function is problem-specific, and can be based on biological knowledge or empirical data. Given sequences $Q = \{s_1, s_2, s_3\}$ where $s_1 = TTA, s_2 = GC, s_3 = AC$, we can construct an alignment $A$ with rows $A_1 = TT{-}{-}A, A_2 = {-}{-}G{-}C, A_3 = A{-}{-}C{-}$, shown in Figure 2. Formally, the alignment costs of two rows in an alignment, respectively the full alignment, are defined as follows.

**Definition 1** (Pair Score)**.** *Let $s_i, s_j \in Q$, with $i \neq j$, be two sequences over alphabet $\Sigma$, and $A$ an alignment of $Q$. Then the pair score of $s_i$ and $s_j$ in $A$ is $C(A, i, j) = \sum_{k=1}^{m} sub(a_{ik}, a_{jk})$.*

**Definition 2** (Alignment Score)**.** *The* alignment score*, or* sum of pairs score *of alignment $A$ is $C(A) = \sum_{1 \leq i < j \leq n} C(A, i, j)$.*

We also define a score function for an alignment *column*.

**Definition 3** (Column Score)**.** *The* column score *of a column $\mathcal{C}$ of alignment $A$ is defined as $C(\mathcal{C}) = \sum_{1 \leq i < j \leq n} sub(\mathcal{C}_i, \mathcal{C}_j)$, where $\mathcal{C}_i$ and $\mathcal{C}_j$ denote the characters in rows $i$ and $j$ of column $\mathcal{C}$.*

**Example 1.** *Consider sequences $Q$ and alignment $A$ from Figure 2. The column score of the first column of $A$ is $C(\mathcal{C}_1) = sub(T, -) + sub(T, A) + sub(-, A)$.*

The goal of MSA is to find an optimal alignment $A$, i.e., one where for all alignments $A' : C(A) \leq C(A')$.

### Solving MSA Tasks with State-Space Search

MSA tasks can be solved via state-space search in a transition system, defined as $\mathcal{T} = \langle \mathcal{S}, L, T, cost, I, G \rangle$ where $\mathcal{S}$ is a set of states, $L$ is a set of labels, $T \subseteq \mathcal{S} \times L \times \mathcal{S}$ is a set of transitions, $cost : L \to \mathbb{N}_0^+$ is a cost function assigning non-negative costs to labels, $I \in \mathcal{S}$ is the initial state, and $G \subseteq \mathcal{S}$ is a set of goal states. Solving the MSA problem then corresponds to finding a lowest-cost path from the initial state to a goal state in $\mathcal{T}$.

Given a set of sequences $\{s_1, \dots s_n\}$, we obtain the corresponding MSA transition system $\mathcal{T}$ as follows. The states $\mathcal{S}(\mathcal{T})$ are the vertices in an $n$-dimensional lattice graph $\mathcal{G}$ of size $(|s_1| + 1) \times \cdots \times (|s_n| + 1)$. The initial state $I(\mathcal{T})$ is at the origin $\langle 0, \dots, 0 \rangle$, and the single goal state $g$ is at coordinate $\langle |s_1|, \dots, |s_n| \rangle$. There is a transition $s \xrightarrow{\ell} s'$ between state $s = \langle x_1, x_2, \dots, x_n \rangle$ and state $s' = \langle x'_1, x'_2, \dots, x'_n \rangle$ if, and only if, between the two states there is at most a single *step of progress*—advancing a coordinate $x_i$ by one, which consumes the next character of sequence $s_i$—in every dimension, and in at least one dimension one step of progress is made. Formally, this is the case if $0 \leq x'_i - x_i \leq 1$ for all $i = 1 \dots n$, and $\sum_{i=1}^{n} (x'_i - x_i) > 0$.

A transition $s \xrightarrow{\ell} s'$ corresponds to a column in the alignment, where the characters in the column are determined by the dimensions in which progress is made between $s$ and $s'$. The cost of a transition is equal to the column score of the corresponding column in the alignment, which is the sum of the substitution costs across all pairs of characters in that column. This kind of cost structure is referred to as *factored transition costs*, where the cost of a transition is determined by its factors. In MSA, we model these factors using *cost components*, each representing the interaction of a specific pair of sequences. The cost of a cost component is the substitution cost of the corresponding two characters in the alignment. A *cost component class* is the set of all cost components defined over the same pair of sequences.

### Admissible Heuristics for MSA

A *heuristic* for MSA is a function $h : \mathcal{S} \to \mathbb{N}_0$ that estimates goal distances. We refer to a heuristic as *admissible* if it never overestimates the true goal distance of any state, and as *perfect* if it always equals the true goal distance. We let $h(cost, s)$ denote the heuristic value of state $s$ under cost function $cost$. $A^*$ search (Hart, Nilsson, and Raphael 1968)

is guaranteed to find an optimal solution when using an admissible heuristic.

*Abstraction heuristics* use an abstraction function $\alpha$ to create a coarser representation $\mathcal{T}' = \alpha(\mathcal{T})$ of the state space. For MSA, we follow Skjelnes, Gnad, and Seipp (2024) and use *patterns*: a *pattern* $P \subseteq Q$ is a subset of the input sequences that induces a projection onto the corresponding sub-alignment problem (Culberson and Schaeffer 1998). The resulting abstract state space has the property that the optimal goal distance of any abstract state is a lower bound on the optimal goal distance of the corresponding concrete states. Two MSA patterns $P$ and $P'$ *conflict* if they share at least one cost component. A pattern collection $\mathcal{P}$, a set of patterns, is *strictly conflicting* if all patterns in $\mathcal{P}$ share at least one common cost component. We denote the abstract transition system induced by pattern $P$ as $\mathcal{T}_P$, and the corresponding abstraction heuristic as $h^P$. We refer to a pattern $P$ as a *k*D *pattern* if it contains exactly $k$ sequences. We say that a pattern heuristic $h^P$ is *relevant* to a cost component class $c^{i,j}$ (or to a cost component $c \in c^{i,j}$) if $P$ contains both sequences $s_i$ and $s_j$.

Two prominent admissible MSA heuristics are the *all-k* and *one-k* heuristics (Kobayashi and Imai 1998). Given $n$ sequences, the all-k heuristic is computed as the sum of all abstraction heuristics induced by patterns of size $k$, divided by $\binom{n-2}{k-2}$ to ensure admissibility. The one-k heuristic sums one $k$D pattern heuristic, one $(n-k)$D pattern heuristic and all 2D patterns that include one sequence from each of the two patterns.

A *cost partitioning* over a set of admissible heuristics $\mathcal{H}$ maps each heuristic $h \in \mathcal{H}$ to a cost function $cost_h$, such that $\sum_{h \in \mathcal{H}} cost_h(\ell) \leq cost(\ell)$ for all labels $\ell \in L$. The resulting *cost-partitioned heuristic* $\sum_{h \in \mathcal{H}} cost_h(\ell)$ is guaranteed to be admissible. *Post-hoc optimization* (PhO) is a cost partitioning algorithm that optimizes weights assigned to each heuristic with a linear program (LP), such that the overall cost-partitioned heuristic is maximized while remaining admissible. The *MSA PhO* is a variant of PhO that accounts for the maximal strictly conflicting pattern collections (Skjelnes, Gnad, and Seipp 2024). A *maximal strictly conflicting* pattern collection $\mathcal{P}'$ consists of all patterns that include a specific cost component class $c^{i,j}$; by design, all patterns in $\mathcal{P}'$ share the full cost component class $c^{i,j}$, and there is one such collection per cost component class.

**Definition 4.** *(Skjelnes, Gnad, and Seipp 2024) The MSA PhO linear program is defined as follows:*

$$h^{\mathcal{P}}_{\text{PhO}}(s) = \textit{maximize} \quad \sum_{i=1}^{n} w_i \cdot h^{P_i}(s) \quad s.t.$$

$$\sum_{P_i \in \mathcal{P}'} w_i \leq 1 \textit{ for all maximal strictly conflicting } \mathcal{P}' \subseteq \mathcal{P}$$

$$w_i \geq 0 \quad \textit{for all } P_i \in \mathcal{P}$$

*Uniform cost partitioning* (UCP) divides the cost of each label equally among all heuristics that use it. In MSA, UCP is a feasible, but not necessarily optimal, solution to the MSA PhO linear program (Skjelnes, Gnad, and Seipp 2024). Moreover, UCP over all sequence subsets of size $k$ is equivalent to the *all-k* heuristic (Kobayashi and Imai 1998; Skjelnes, Gnad, and Seipp 2024).

Some cost partitioning algorithms rely on an ordering $\omega$ of the abstractions to assign costs. *Greedy zero-one cost partitioning* (GZOCP) is one such algorithm, which assigns the full cost to the first appearance of each label $\ell$ in the abstraction order defined by $\omega$. *Saturated cost partitioning* (SCP) is another ordering-based cost partitioning algorithm, which extends GZOCP by preserving unneeded costs for later abstractions in the ordering. SCP computes a *saturated cost function* for each abstraction in the ordering, which is a reduced cost function that does not decrease the heuristic value for that abstraction. For an abstraction heuristic $h^\alpha$ using abstract transition system $\mathcal{T}^\alpha$, we can compute the unique *minimum saturated cost function* $saturate(h^\alpha, cost) = mscf$ as follows (Seipp and Helmert 2018):

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

Here, $h^*_\alpha$ is the perfect heuristic for $\mathcal{T}^\alpha$, and $\ominus$ denotes regular subtraction, except that $\infty - \infty = -\infty$. The saturated cost partitioning $\langle cost_1, \ldots, cost_n \rangle$ is computed as

$$\begin{aligned} remain_0 &= cost \\ cost_i &= saturate(h_i, remain_{i-1}) && \text{for all } 1 \leq i \leq n \\ remain_i &= remain_{i-1} - cost_i && \text{for all } 1 \leq i \leq n \end{aligned}$$

where $remain_i$ is the remaining cost function after processing the first $i$ abstractions in the ordering $\omega$, and $cost_i$ is the saturated cost function for the $i$-th abstraction in $\omega$. Figure 3 illustrates the concept of saturated costs for a single abstraction.

**Example 2.** *Consider Figure 3, showing a transition system with five labels $\ell_1$ to $\ell_5$. The saturated cost for label $\ell_5$ is 6, which is less than its original cost of 10. This means that we can save* $10 - 6 = 4$ remaining cost *for $\ell_5$ in this transition system without affecting the optimal solution cost.*

*Saturated post-hoc optimization* (SPhO) does not rely on an ordering of the abstractions, but uses saturated cost functions to tighten constraints of the PhO LP (Seipp, Keller, and Helmert 2021).

**Definition 5.** *The SPhO LP is defined as follows:*

$$\begin{aligned} \textit{maximize} \quad & \textstyle\sum_{h \in \mathcal{H}} h(cost, s) \cdot w_h \quad s.t. \\ \textstyle\sum_{h \in \mathcal{H}} scf_h(\ell) \cdot w_h \quad & \leq cost(\ell) \quad \textit{for all labels } \ell \in L \\ w_h \quad & \geq 0 \quad \textit{for all } h \in \mathcal{H} \end{aligned}$$

We say that heuristic $h_1$ *dominates* $h_2$ if $h_1(s) \geq h_2(s)$ for all states $s$. In classical planning, SCP dominates GZOCP (Seipp, Keller, and Helmert 2017) and SPhO dominates PhO. In both cases, there are examples of states for which the dominated algorithm produces strictly worse estimates. SCP and SPhO are incomparable, i.e., there exists no dominance relation between them (Seipp, Keller, and Helmert 2021).

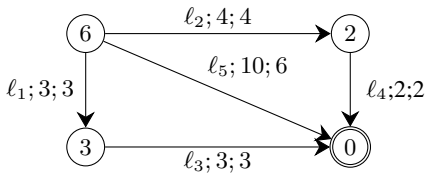



Figure 3: A transition system where transitions are annotated with their label $\ell_i$, original cost $cost(\ell_i)$ and saturated cost $scf(\ell_i)$, separated by semicolons. States are annotated with goal distance. The remaining cost for each label can be computed as $cost(\ell_i) - scf(\ell_i)$.

## Greedy Zero-One Cost Partitioning in MSA

We first investigate GZOCP for MSA (Riesterer 2018). Generalizing GZOCP to the factored transition costs of MSA is straightforward, since we can simply assign the full (or zero) cost to the corresponding cost components. We show that, despite this simplicity, GZOCP in MSA dominates the specialized $h^{\text{all},2}$ and $h^{\text{one},3}$ heuristics. We build on the following known result.

**Lemma 1.** *(Carrillo and Lipman 1988)* $h^{\{s_i,s_j,s_k\}} \geq h^{\{s_i,s_j\}} + h^{\{s_i,s_k\}} + h^{\{s_j,s_k\}}$.

GZOCP may produce heuristics where one or more cost component classes have zero cost. For a heuristic on pattern $\{s_i, s_j, s_k\}$ in which only the cost component classes in a set $C$ have non-zero cost, we write $h_C^{\{s_i,s_j,s_k\}}$, and denote the corresponding transition system by $\mathcal{T}_{\{s_i,s_j,s_k\}_{\{C\}}}$.

A 3D pattern heuristic in which only the cost component classes $\{c^{i,j}\}$, respectively $\{c^{i,j}, c^{i,k}\}$, are non-zero does not collapse to zero: by projecting optimal 3D goal paths onto the corresponding 2D transition systems, it can be shown that $h^{\{s_i,s_j,s_k\}}_{\{c^{i,j}\}} = h^{\{s_i,s_j\}}$ and $h^{\{s_i,s_j,s_k\}}_{\{c^{i,j},c^{i,k}\}} \geq h^{\{s_i,s_j\}} + h^{\{s_i,s_k\}}$.

The main observation is that goal paths in a 3D transition system for $\{s_i, s_j, s_k\}$ projects to goal paths in its 2D counterparts $\{s_i, s_j\}$, $\{s_i, s_k\}$ and $\{s_j, s_k\}$, but the converse is not necessarily true: a combination of optimal goal paths in the 2D transition systems may not form a valid goal path in the 3D transition system.

**Theorem 2.** *Let $\mathcal{P}$ be a collection of 2D and 3D patterns, and let $\mathcal{P}_{\text{all},2}$ be the collection of all 2D patterns. If $\mathcal{P} \supseteq \mathcal{P}_{\text{all},2}$, then $h^{\text{GZOCP}}_{\mathcal{P}}$ dominates $h^{\text{all},2}$ for any pattern ordering $\omega$.*

*Proof.* For any ordering $\omega$, GZOCP assigns the full cost of each cost component class to the first pattern in $\omega$ that contains it, and zero cost to every later pattern; no admissibility normalization is needed. This assigns each class to exactly one pattern, partitioning the classes by the pattern that receives them. For a pattern that receives at least one class, its contribution to $h^{\text{GZOCP}}_{\mathcal{P}}$ is the heuristic of that pattern restricted to its assigned classes, and this contribution dominates the sum of the 2D heuristics of the same classes: a 2D pattern carries its single class and contributes exactly $h^{\{s_i,s_j\}}$; a 3D pattern $\{s_i, s_j, s_k\}$ that carries one or two classes satisfies $h^{\{s_i,s_j,s_k\}}_{\{c^{i,j}\}} = h^{\{s_i,s_j\}}$, respectively $h^{\{s_i,s_j,s_k\}}_{\{c^{i,j},c^{i,k}\}} \geq h^{\{s_i,s_j\}} + h^{\{s_i,s_k\}}$, by the observation above; and a 3D pattern that carries all three classes satisfies $h^{\{s_i,s_j,s_k\}} \geq h^{\{s_i,s_j\}} + h^{\{s_i,s_k\}} + h^{\{s_j,s_k\}}$ by Lemma 1. Since $\mathcal{P} \supseteq \mathcal{P}_{\text{all},2}$, every class is assigned to some pattern, so summing over all patterns yields $h^{\text{GZOCP}}_{\mathcal{P}} \geq \sum_{1\leq i<j\leq n} h^{\{s_i,s_j\}} = h^{\text{all},2}$. □

GZOCP can also emulate and even dominate $h^{\text{one},3}$ based on the selection of $\omega$.

**Theorem 3.** *Let $\mathcal{P}$ contain the patterns used by $h^{\text{one},3}$. Then there exists an ordering $\omega$ such that $h^{\text{GZOCP}}_{\mathcal{P}}$ dominates $h^{\text{one},3}$.*

*Proof.* $h^{\text{one},3}$ sums one 3D pattern heuristic, one $(n-3)$D pattern heuristic, and all 2D patterns that bridge the two, assigning full cost to each. We consider the computable cases $n \leq 6$: for $n < 6$ the second pattern has at most two sequences, for $n = 6$ it is a second 3D pattern, and for $n > 6$ it has more than three sequences and is outside the 2D/3D patterns we compute. Let $\omega$ process the one (or, for $n = 6$, two) 3D patterns of $h^{\text{one},3}$ first, followed by all 2D patterns. GZOCP then assigns the full cost of each class internal to a 3D pattern to that pattern, and the full cost of each remaining bridging class to its 2D pattern, while the 2D patterns internal to a 3D pattern receive zero cost. This reproduces the cost assignment of $h^{\text{one},3}$ exactly, so $h^{\text{GZOCP}}_{\mathcal{P}} \geq h^{\text{one},3}$, with equality when $\mathcal{P}$ holds only these patterns and strict dominance possible when it holds more. □

This shows that GZOCP is more flexible than $h^{\text{all},2}$ and $h^{\text{one},3}$: equivalent when using the same patterns, and possibly dominating both for general pattern collections.

## Cost Saturation for MSA

To adapt SCP and SPhO to MSA, we need to define how to saturate factored transition costs and how to compute the remaining cost of cost components after saturation.

### Ambiguities from Factored Transition Costs

Computing the saturated costs over the transition labels is straightforward. For each transition $t = s \xrightarrow{\ell} s'$ in $\mathcal{T}_P$ of pattern $P$, it is computed as $scf(\ell) = h^*_P(s') - h^*_P(s)$. The remaining cost of a transition label $\ell$ is $rem(\ell) = cost(\ell) - scf(\ell)$. Computing the remaining costs of the individual cost components that constitute a transition label $\ell$ is more involved (see Figure 5 for an illustration). Usually, there are multiple cost components that contribute to the same transition label so $rem(\ell)$ needs to be distributed across these cost components. How exactly that distribution is done is important for patterns that occur later in the SCP ordering which share the same cost components, since their heuristic value depends on the distribution. Thus, there is ambiguity in how much remaining cost each cost component should receive. We call this the *cost ambiguity problem*, which we model as a *remaining cost constraint*: $\sum_{c\in\ell} rem(c) \leq rem(\ell)$. Collecting these constraints for all labels results in a system of

linear inequalities. We refer to the problem of finding a solution that maximizes the final heuristic value after SCP as the *cost redistribution problem*.

The main issue is that there is no simple recipe that optimizes the heuristic value: maximizing the total redistributed cost does not necessarily lead to the best heuristic value, as illustrated in Figure 4. This is unlike classical planning, where each label has a single cost and a remaining cost constraint has the form $rem(\ell) \leq cost(\ell)$, so the optimal redistribution simply assigns the maximum value that satisfies each constraint.

We next show that the redistribution can be based on the pattern ordering $\omega$. This works only if no two cost components that share a remaining cost constraint also appear together in more than one heuristic, unlike the example in Figure 4. This condition indeed holds for 2D and 3D patterns, which we establish next.

**Proposition 4.** *Let $\mathcal{P}$ be a pattern collection with $|P| \leq 3$ for all $P \in \mathcal{P}$, and let $P$, $P' \in \mathcal{P}$, with $P \neq P'$, be two patterns. If there exist $c_1 \in c^{i,j}$ and $c_2 \in c^{i,k}$, where $j \neq k$, that are both part of a label $l$ in $\mathcal{T}_P$, $\{c_1, c_2\} \subseteq l$, then there does not exist a label $l'$ in $\mathcal{T}_{P'}$ such that $\{c_1, c_2\} \subseteq l'$.*

*Proof.* Assume for contradiction that both cost components $c_1$ and $c_2$ conflict with $P'$. It follows that $P'$ must include the sequences $s_i$, $s_j$ and $s_k$ to accommodate both cost component classes. However, since $P'$ has at most three sequences, this implies that $P'$ is exactly $\{s_i, s_j, s_k\} = P$, which contradicts the assumption that $P \neq P'$. □

It follows that the case in Figure 4 cannot actually occur when restricting ourselves to patterns of at most three sequences, as at most one of $c_1$ and $c_2$ can appear in any transition system $\mathcal{T}_{P'}$. This means that the pattern ordering can be used to disambiguate the redistribution problem, by lazily assigning the maximum possible cost to the cost component that appears in the next pattern in $\omega$.

**Lemma 5.** *A pattern ordering $\omega$ over a pattern collection $\mathcal{P}$ with $|P| \leq 3$ for all $P \in \mathcal{P}$ imposes a non-ambiguous cost redistribution order over the cost components.*

*Proof.* From Proposition 4, no two cost components that subscribe to the same remaining cost constraint can appear together in more than one pattern. Thus, by the order imposed by $\omega$, for each pattern $P \in \mathcal{P}$, when redistributing costs in $P$, at most one cost component per remaining cost constraint will conflict with $P$. □

Next we will describe how to efficiently compute a redistribution for a cost component. Instead of considering all remaining cost constraints that involve the cost component $c$, we show that only the set of constraints formed from the most recently processed conflicting pattern $P$ whose induced transition system $\mathcal{T}_P$ has been solved and saturated need to be considered.

When redistributing costs to a given pattern, the algorithm only needs to look at the constraints coming from the latest previously processed pattern that conflicts with the same cost component, simplifying the computation. We exploit this in the concept of newly formed constraints.

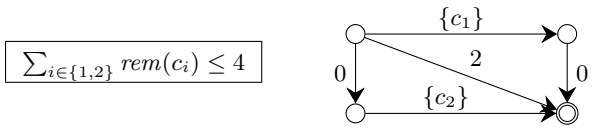



Figure 4: Remaining costs from a previous abstraction that are to be distributed to cost components $c_1$ and $c_2$ in a transition system in which both appear.

**Definition 6** (Newly-Formed Constraint)**.** *Given a pattern collection $\mathcal{P}$ and a pattern ordering $\omega$, the* newly formed constraints *for a cost component $c$, denoted $NC(c)$, is the set of constraints of the most recently processed pattern $P_i$ (with respect to $\omega$) that conflicts with $c$. The newly formed constraints for a cost component class $c^{i,j}$ is denoted $NC(c_{i,j})$.*

**Example 3.** *In Figure 6, pattern $P = \{s_1, s_2, s_4\}$ is the next pattern to be processed. The arrows indicate the most recently processed pattern for each cost component class that conflicts with $P$. For instance, the cost component class $c^{1,2}$ conflicts with $P$, and the most recently processed conflicting pattern is $P' = \{s_1, s_2, s_3\}$. Thus, $NC(c)$ for each cost component $c \in c^{1,2}$ are the remaining cost constraints formed from $P'$ that involve $c$. On the other hand, the cost component class $c^{2,4}$ also conflicts with $P$, but it does not conflict with any previously processed pattern. Thus, $NC(c)$ for each cost component $c \in c^{2,4}$ is the original cost of $c$, $cost(c)$.*

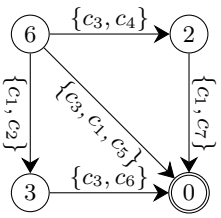
**Proposition 6.** *If redistribution is greedy, then the constraints of $NC(c)$ are sufficient to determine the redistribution for cost component $c$: any other cost constraint that $c$ subscribes to outside of $NC(c)$ is redundant.*

*Proof.* By definition, $NC(c)$ is the set of remaining cost constraints that have been formed from the latest processed pattern that conflicts with $c$, and greedy redistribution makes any earlier constraint involving $c$ redundant. □

The key insight is that the newly formed constraints are sufficient to determine the next redistribution. And since all newly formed constraints for a cost component $c$ in class $c^{i,j}$ originate from the same pattern, we only have to keep track of the pattern that is the latest one for each cost component. While we still maintain a redistributed cost for each cost component, this reduces the number of patterns that need to be inspected when determining a redistribution from $O(n)$ to 1, where $n$ is the number of patterns.

## Saturated Cost Partitioning for MSA

We use the simplifications presented in the previous sections to design a saturated cost partitioning algorithm for MSA, presented in Algorithm 1. It processes patterns in the given ordering $\omega$. For each pattern $P_i$, we (i) initialize the induced transition system $\mathcal{T}_{P_i}$ with zero transition costs (line 2), (ii) assign each cost component $c$ a redistributed cost based on its newly formed constraints (line 4), (iii) solve and saturate $\mathcal{T}_{P_i}$ to obtain the remaining cost constraints $remain_i$ (lines 5–6), and (iv) update the newly formed constraints $NC(c)$ for all cost components in $\mathcal{T}_{P_i}$ (line 8). Here,




$$\sum_{i\in\{3,4\}} rem(c_i) \leq 0$$
$$\sum_{i\in\{1,2\}} rem(c_i) \leq 0$$
$$\sum_{i\in\{3,6\}} rem(c_i) \leq 0$$
$$\sum_{i\in\{1,7\}} rem(c_i) \leq 0$$
$$\sum_{i\in\{3,1,5\}} rem(c_i) \leq 4$$

Figure 5: Left: transition system with cost components labeling transitions; right: the corresponding remaining costs associated with each label collection of cost components.

**Algorithm 1** Saturated Cost Partitioning for MSA.

**Require:** Pattern collection $\mathcal{P}$, pattern ordering $\omega$
**for all** patterns $P_i \in \mathcal{P}$ in the order $\omega$ **do**
  INITIALIZE($\mathcal{T}_{P_i}$)
  **for all** cost components $c$ occurring in $\mathcal{T}_{P_i}$ **do**
    $cost_i(c) \leftarrow$ REDISTRIBUTE($c$)
  SOLVE($\mathcal{T}_{P_i}$)
  $remain_i \leftarrow saturate(h^{P_i}, cost_i)$
  **for all** cost components $c$ occurring in $\mathcal{T}_{P_i}$ **do**
    $NC(c) \leftarrow \{\, \gamma \in remain_i \mid c \in \gamma \,\}$

**function** REDISTRIBUTE($c$)
  **if** $NC(c) = \emptyset$ **then**
    **return** $cost(c)$
  **return** $\min_{\gamma \in NC(c)} \gamma.val$

$remain_i$ is a set of remaining cost constraints; each constraint $\gamma$ bounds a sum of cost component costs, and $\gamma.val$ denotes its right-hand-side value.

## Saturated Post-hoc Optimization for MSA

In many classical planning benchmark domains, saturated post-hoc optimization (SPhO) yields significantly stronger heuristics than post-hoc optimization (PhO) (Seipp, Keller, and Helmert 2021). It does so by strengthening the constraints of the PhO linear program through cost saturation. One might expect such results to carry over to MSA, but in fact, we show that in MSA, SPhO is equivalent to PhO.

### SPhO Linear Program for MSA

Definition 5 gives the classical-planning SPhO LP with per-label constraints $\sum_{h\in\mathcal{H}} scf_h(\ell) \cdot w_h \leq cost(\ell)$. In MSA, label costs factor into pairwise cost components, so admissibility must hold for each component separately. We therefore replace labels $\ell$ by components $c$ and interpret $scf_h(c)$ as the saturated contribution of component $c$ to $h$, yielding the MSA-specific LP in Definition 7.

**Definition 7.** *Given a set of heuristics $\mathcal{H}$ for an MSA problem, the* SPhO linear program for MSA *is defined as follows:*

$$\textit{maximize} \sum_{h\in\mathcal{H}} h(cost, s) \cdot w_h \quad \textit{subject to}$$
$$\sum_{h\in\mathcal{H}} scf_h(c) \cdot w_h \leq cost(c) \textit{ for all cost components } c$$
$$w_h \geq 0 \textit{ for all } h \in \mathcal{H}$$

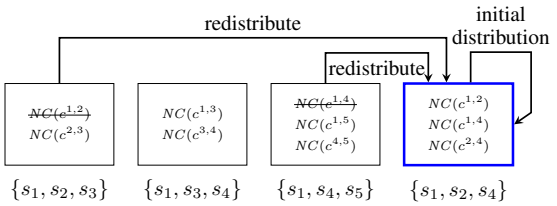



Figure 6: Redistribution resolves refunding for each cost component $c$ based on newly formed constraints $NC(c)$.

There are two differences between this MSA SPhO LP and its PhO counterpart from Definition 4: First, the SPhO LP uses the saturated cost $scf_h(c)$ in heuristic $h$ instead of the original cost $cost(c)$, where $0 \leq scf_h(c) \leq cost(c)$. Second, the SPhO LP defines a constraint for each cost component $c$, whereas the PhO LP defines a constraint per maximal strictly conflicting set of patterns. Maximal strictly conflicting pattern sets correspond exactly to the cost component classes they share. This means if one cost component $c$ in a cost component class $c^{i,j}$ has $scf_h(c) = cost(c)$ for all heuristics $h$ that are relevant to $c$, then we can rewrite the SPhO constraint for $c$ as $\sum_{h\in\mathcal{H}} cost(c) \cdot w_h \leq cost(c)$, which is equal to the PhO constraint for the cost component class $c^{i,j}$. This constraint dominates any other constraint for a cost component in the same class. To illustrate this, we can rewrite

$$\sum_{h\in\mathcal{H}} cost(c) \cdot w_h \leq cost(c)$$

as

$$\sum_{h_i\in\mathcal{H}} 1 \cdot w_{h_i} \leq 1 \tag{1}$$

Any other constraint for a cost component $c'$ in the same class will have the form

$$\sum_{h_i\in\mathcal{H}} a_i \cdot w_{h_i} \leq 1 \tag{2}$$

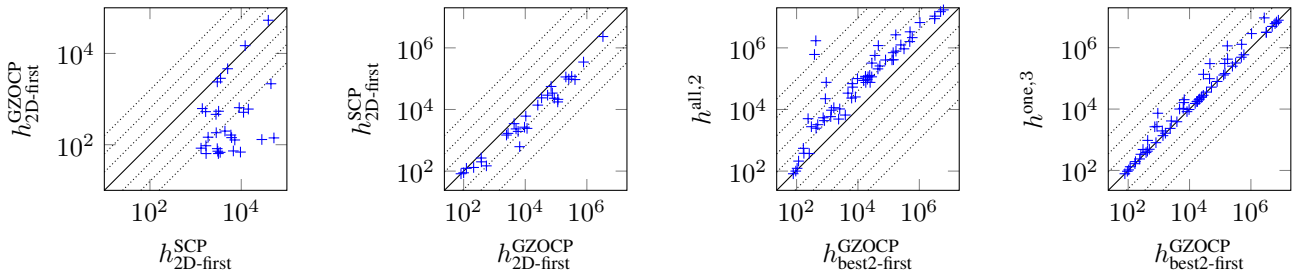
where $a_i = \frac{scf_{h_i}(c')}{cost(c')}$. Since $0 \leq \frac{scf_{h_i}(c')}{cost(c')} \leq 1$, we have that (1) dominates (2).

We will next show that for each cost component class $c^{i,j}$, there is at least one cost component $c$ such that $scf_h(c) = cost(c)$ for all heuristics $h$ that include $c$.

**Lemma 7.** *If a transition $t = s \xrightarrow{\ell} s'$ is on an optimal goal path from $s$ to a goal state in the abstract transition system, then its saturated cost is equal to its cost.*

**Lemma 8.** *Each cost component class $c^{i,j}$ contains at least one cost component $c$ such that $scf_h(c) = cost(c)$ for all heuristics $h$ that $c$ is relevant for.*

*Proof.* Consider the cost component $c = c^{i,j}_{\langle |s_i|-1, |s_j| \rangle \to \langle |s_i|, |s_j| \rangle}$. This cost component will be part of a label $\ell$ of a transition $t$ that only makes progress on sequence $s_i$ from state $s$ to a goal state $g$, for every heuristic $h$ that $c$ is relevant for. This is because any heuristic that is relevant for $c$ must include sequences $s_i$ and $s_j$ in its




(a) GZOCP vs SCP: time (b) GZOCP vs SCP: nodes (c) GZOCP vs $h^{\text{all},2}$: nodes (d) GZOCP vs $h^{\text{one},3}$: nodes

Figure 7: Selected pairwise comparisons of runtime and node expansions. The solid diagonal marks parity ($y = x$); the dotted lines mark progressively larger ratios between the two configurations.

corresponding pattern. The transition $t$ is the only transition from $s$ to $g$, so it must be on an optimal goal path from $s$ to $g$. From Lemma 7 we know that $scf_h(\ell) = cost(\ell)$, which means that $scf_h(c) = cost(c)$. □

**Theorem 9.** *In MSA,* $h^{\text{PhO}} = h^{\text{SPhO}}$.

*Proof.* From Lemma 8 we get $scf_h(c) = cost(c)$ for every heuristic $h$ that $c$ is relevant for. It entails that the SPhO constraint for $c$ is $\sum_{h \in \mathcal{H}} cost(c) \cdot w_h \leq cost(c)$, which is the same as the PhO constraint for the cost component class $c^{i,j}$. We also know that this constraint dominates any other constraint for a cost component in the same class, so we can ignore those other constraints without changing the solution of the SPhO LP. The remaining constraints of the SPhO LP are exactly that of the PhO LP.

□

## Experiments

We implement all cost partitioning algorithms in the MSA-Solver framework (Hatem, Burns, and Ruml 2018) and run our experiments with the Lab toolkit (Seipp et al. 2017). Each task runs on an AMD Ryzen 7 PRO 5850U CPU with a 300-second time limit and 8 GiB memory limit. We evaluate the heuristics on the easy BaliBase benchmark set (Thompson, Plewniak, and Poch 1999).

**Impact of Cost Redistribution.** First, we explore the impact of redistribution by comparing GZOCP with SCP under four pattern orderings: 2D-first (2D patterns first), 3D-first (3D patterns first), best1-first (the 3D pattern with the highest initial-state heuristic value first, then all 2D patterns), and best2-first (the highest-valued 3D pattern and the best 3D pattern that does not conflict with it first, then all 2D patterns). Table 1 compares node expansions between GZOCP and SCP under the same pattern orderings. While SCP can yield a stronger heuristic in cases where redistribution helps, it typically does not pay off in terms of total solution time. Figures 7a and 7b show that this holds even for the ordering most favorable to redistribution, *2D-first*. The main reason for the weak performance of SCP is that the overhead of redistribution does not empirically pay off in terms of total runtime, even in cases where it does reduce search effort. GZOCP only needs to check the redistribution condition once per cost component class $c^{i,j}$ in a pattern. And its redistribution condition is relatively simple, which is whether there exist a previously processed pattern that includes $c^{i,j}$. In contrast, SCP needs to check the redistribution condition for each individual cost component $c$, and the redistribution condition is more complex as it involves resolving the newly formed constraints $NC(c)$.

| | | $h^{\text{GZOCP}}$ | | | | $h^{\text{SCP}}$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 3D-first | 2D-first | best1-first | best2-first | 3D-first | 2D-first | best1-first | best2-first | Solved |
| $h^{\text{GZOCP}}$ | 3D-first | – | 56 | 13 | 8 | **0** | 0 | 5 | 4 | 80.0% |
| | 2D-first | 0 | – | 0 | 0 | 0 | **0** | 0 | 0 | 80.0% |
| | best1-first | 36 | 60 | – | 0 | 19 | 19 | **0** | 0 | 85.3% |
| | best2-first | 42 | 59 | 23 | – | 24 | 24 | 11 | **0** | 85.3% |
| $h^{\text{SCP}}$ | 3D-first | **0** | 28 | 8 | 4 | – | 0 | 5 | 4 | 40.0% |
| | 2D-first | 0 | **28** | 8 | 4 | 0 | – | 5 | 4 | 40.0% |
| | best1-first | 21 | 30 | **7** | 3 | 21 | 21 | – | 3 | 40.0% |
| | best2-first | 24 | 30 | 14 | **0** | 24 | 24 | 11 | – | 40.0% |

Table 1: Pairwise comparison of node expansions between different heuristics. Each cell $(i, j)$ indicates the number of tasks solved by both algorithms where algorithm $i$ needs fewer expansions than algorithm $j$. Bold values indicate comparisons between configurations that use the same pattern ordering.

**Comparison to Baselines.** We compare $h^{\text{all},2}$, $h^{\text{all},3}$, $h^{\text{one},3}$, $h^{\text{SCP}}$ and $h^{\text{GZOCP}}$ with different pattern orderings. We also include three variants of $h^{\text{PhO}}$: $h^{\text{PhO}}_0$ solves the LP once for the initial state, $h^{\text{PhO}}_1$ solves the LP for each evaluated state and $h^{\text{PhO}}_{10\text{k}}$ solves the LP for every 10 000$^{\text{th}}$ state. Table 2 reports a pairwise comparison of *total* node expansions. Total expansions include the final $f$-layer, whose size depends on A$^*$ tie-breaking rather than on heuristic strength, so heuristics in a value-dominance relation can still trade places on individual tasks.

**Heuristic Estimates of PhO.** By construction, $h^{\text{PhO}}_0 \leq h^{\text{PhO}}_{10\text{k}} \leq h^{\text{PhO}}_1$ in heuristic value, as recomputing the LP at

| | | $h^{\text{GZOCP}}$ | | $h^{\text{SCP}}$ | $h^{\text{PhO}}$ | | | Baselines | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 2D-first | best2-first | best2-first | $K$=0 | $K$=1 | $K$=10k | $h^{\text{one},3}$ | $h^{\text{all},2}$ | $h^{\text{all},3}$ | Solved |
| $h^{\text{GZOCP}}$ | 2D-first | – | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80.0% |
| | best2-first | 59 | – | 0 | 0 | 0 | 0 | 23 | 59 | 3 | 85.3% |
| $h^{\text{SCP}}$ | best2-first | 30 | 0 | – | 0 | 0 | 0 | 14 | 30 | 1 | 40.0% |
| $h^{\text{PhO}}$ | $K$=0 | 57 | 58 | 29 | – | 1 | 0 | 57 | 57 | 20 | 88.0% |
| | $K$=1 | 57 | 56 | 29 | 39 | – | 39 | 56 | 57 | 41 | 78.7% |
| | $K$=10k | 57 | 58 | 29 | 6 | 1 | – | 57 | 57 | 24 | 88.0% |
| Baselines | $h^{\text{one},3}$ | 60 | 0 | 0 | 0 | 0 | 0 | – | 60 | 2 | 85.3% |
| | $h^{\text{all},2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | – | 0 | 80.0% |
| | $h^{\text{all},3}$ | 57 | 57 | 29 | 7 | 1 | 3 | 57 | 57 | – | 88.0% |

Table 2: Pairwise comparison of node expansions between different heuristics. Each cell $(i, j)$ indicates the number of tasks solved by both algorithms and in which algorithm $i$ outperformed algorithm $j$.

more states can only tighten the estimate. Moreover, $h^{\text{all},3}$ is a feasible solution of the PhO LP when the LP ranges over all 3D heuristics, so the per-state optimum $h_1^{\text{PhO}}$ dominates $h^{\text{all},3}$ in value, while $h_0^{\text{PhO}}$ and $h_{10\text{k}}^{\text{PhO}}$ only guarantee this relation for states they compute the PhO LP. Table 2 bears out these value relations: $h_1^{\text{PhO}}$ needs no more expansions than any other variant on every task but one, which is a tie-breaking artifact in the final $f$-layer. Recomputing the LP at every state is costly, however, so $h_1^{\text{PhO}}$ solves fewer tasks overall, and its stronger estimates do not improve coverage.

## Conclusions

We study cost partitioning for MSA under factored transition costs, using cost components to adapt the saturation-based methods SCP and SPhO. Theoretically, we show that $h^{\text{SPhO}}$ coincides with $h^{\text{PhO}}$ in MSA and that every traditional MSA heuristic is dominated by at least one cost partitioning heuristic. Empirically, our results confirm the strength of cost partitioning, while $h^{\text{SCP}}$ often reduces node expansions compared to $h^{\text{GZOCP}}$ but does not consistently improve runtime.

## Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

## References

Carrillo, H.; and Lipman, D. 1988. The Multiple Sequence Alignment Problem in Biology. *SIAM Journal on Applied Mathematics*, 48(5): 1073–1082.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern Databases. *Computational Intelligence*, 14(3): 318–334.

Edelkamp, S. 2006. Automated Creation of Pattern Database Search Heuristics. In *Proc. MoChArt 2006*, 35–50.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

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

Hatem, M.; Burns, E.; and Ruml, W. 2018. Solving Large Problems with Heuristic Search: General-Purpose Parallel External-Memory Search. *JAIR*, 62: 233–268.

Ikeda, T.; and Imai, H. 1994. Fast A* Algorithms for Multiple Sequence Alignment. *Genome Informatics*, 5: 90–99.

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

Kobayashi, H.; and Imai, H. 1998. Improvement of the A* Algorithm for Multiple Sequence Alignment. *Genome Informatics*, 9: 120–130.

Needleman, S. B.; and Wunsch, C. D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3): 443–453.

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.; Helmert, M.; and Bonet, B. 2014. LP-based Heuristics for Cost-optimal Planning. In *Proc. ICAPS 2014*, 226–234.

Riesterer, M. 2018. *Cost Partitioning Techniques for Multiple Sequence Alignment*. Master's thesis, University of Basel.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *JAIR*, 62: 535–577.

Seipp, J.; Keller, T.; and Helmert, M. 2017. A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning. In *Proc. ICAPS 2017*, 259–268.

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

Seipp, J.; Keller, T.; and Helmert, M. 2021. Saturated Post-hoc Optimization for Classical Planning. In *Proc. AAAI 2021*, 11947–11953.

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

Skjelnes, M.; Gnad, D.; and Seipp, J. 2024. Cost Partitioning for Multiple Sequence Alignment. In *Proc. ECAI 2024*, 4224–4231.

Thompson, J. D.; Plewniak, F.; and Poch, O. 1999. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1): 87–88.

Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A General Theory of Additive State Space Abstractions. *JAIR*, 32: 631–662.