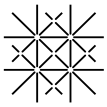


University
of Basel

Cost Partitioning Techniques For Multiple Sequence Alignment

Master thesis

Faculty of Science
Department of Mathematics and Computer Science
Artificial Intelligence
http://ai.cs.unibas.ch/

Examiner: Prof. Dr. Malte Helmert
Supervisor: Silvan Sievers

Mirko Riesterer
mirko.riesterer@stud.unibas.ch
11-064-011

21.08.2018

Acknowledgments

At this point I would like to thank Silvan Sievers for his support and feedback during the course of this thesis. I would also like to thank Prof. Dr. Malte Helmert for allowing me to write this thesis in the artificial intelligence research area. Furthermore I want to thank all my family and friends for their ongoing support during the last months. I also want to thank Timo Enger for his helpful mathematical insight.

Abstract

Multiple Sequence Alignment (MSA) is the problem of aligning multiple biological sequences in the evoluationary most plausible way. It can be viewed as a shortest path problem through an n-dimensional lattice. Because of its large branching factor of $2^n - 1$, it has found broad attention in the artificial intelligence community. Finding a globally optimal solution for more than a few sequences requires sophisticated heuristics and bounding techniques in order to solve the problem in acceptable time and within memory limitations. In this thesis, we show how existing heuristics fall into the category of combining certain pattern databases. We combine arbitrary pattern collections that can be used as heuristic estimates and apply cost partitioning techniques from classical planning for MSA. We implement two of those heuristics for MSA and compare their estimates to the existing heuristics.

Table of Contents

1 Introduction

Multiple Sequence Alignment (MSA) is the problem of aligning multiple biological sequences by adding gaps within the sequences in such a way that corresponding letters of the sequences are ligned up in the same column. In general, correspondence between letters is determined by their biological similarity. MSA is mainly used in computational biology to detect relationships between organisms and how biological sequences of related species changed during evolution.

The quality of an alignment of two sequences is typically judged by a score matrix first introduced by Dayhoff et al. (1978), who created the popular PAM (*point accepted mutation*) matrix. Other commonly used score matrices are the BLOSUM (*blocks substitution matrix*) matrices (Henikoff and Henikoff, 1992). A score matrix assigns costs to each pair of letters (including the cost of aligning a gap letter) in a column of an alignment. Biologically more plausible alignments can be obtained using *affine gap costs*, that does not score gaps in a linear fashion. Instead, extending an already existing gap costs less than opening a new gap.

The *pair score* is the sum of costs over each column. An optimal solution is the pairwise alignment with minimal pair score. For multiple sequences, the pairwise alignments for all possible pairs of sequences are summed (the *sum of pairs score*). The sum of pairs scoring method is a common approach in the literature. Other methods use an iterative approach or progressively align all sequences to one consensus sequence or use a phylogenetic tree to align siblings to their ancestors (Wallace et al., 2006). However, most of the alternative methods only approximate an optimal mathematical solution or find exact solutions only for parts of the sequences. While the sum of pairs scoring method poorly compares in terms of computational complexity and may not be the most accurate method from a biological viewpoint, it can be better used to find mathematically exact solutions. Solving the MSA problem with sum of pairs score has been shown to be NP-complete and its space and time complexity depends exponentially on the number of sequences to align (Wang and Jiang, 1994).

The MSA problem can be formulated as the shortest path problem through an n-dimensional lattice, where each dimension corresponds to one of n sequences. A path through the whole lattice corresponds to an alignment of these sequences. In the two-dimensional case, each move through the lattice corresponds to either adding a gap in one

of the sequences (vertical or horizontal move) or aligning the next letter in both sequences (diagonal move). An optimal solution is the least cost path through the whole lattice. This formulation as a path finding problem makes the MSA problem interesting from an optimal planning perspective. Finding globally optimal solutions to the MSA problem has been an active research topic for the last three decades.

The dynamic programming approach from Needleman and Wunsch (1970) produces optimal alignments for a two-dimensional lattice. Although it can be extended to more dimensions, it only works for at most three sequences in acceptable time.

In order to reduce the exponentially growing space and time complexity with the number of sequences, Carrillo and Lipman (1988) suggested to prune the search space in a certain way. Vertices with costs higher than a certain upper bound do not need to be considered in the search. A simple upper bound would be any path through the whole lattice.

Ikeda and Imai (1994) first make use of the A* algorithm for multiple sequences also using an upper bound. They use an estimator that sums the optimal alignment cost for each pair of sequences. This heuristic is calculated once and can then be applied in every state of the main search. Kobayashi and Imai (1998) improve the accuracy of this estimator by using higher-dimensional optimal sub-alignments.

The best-first search algorithm A* stores child nodes in an open list. For n sequences, MSA has a branching factor of $2^n - 1$. Yoshizumi et al. (2000) presented Partial Expansion A*(PEA*), which does not open the child nodes if they will likely never be expanded and return the parent node with lower priority to the open list. This way, they effectively reduce the memory requirements of the open list.

Schroedl (2005) introduced the Iterative-Deepening Dynamic Programming (IDDP) algorithm, which combines the predetermined search order from dynamic programming with repeated searches with iteratively narrowing bounds. They effectively reduce the number of nodes to be stored. Their algorithm can visit 4 times as many nodes as A*.

Because IDDP can still exhaust the amount of available memory, Hatem and Ruml (2013) propose Parallel External Partial Expansion A*(PE2A*). Their algorithm combines the heuristic estimates from Kobayashi and Imai (1998) with parallel external memory best-first search. They first could solve all instances of the popular BAliBASE Benchmark Reference Set 1 (Thompson et al., 1999) using affine gap costs.

In this thesis, we are concerned with global optimal alignments and use the sum of pairs scoring method. We formalize exisiting heuristics through the concept of *pattern databases* (PDBs). We show, how these heuristics can be generalized to combining arbitrary *non-conflicting pattern collections*. We then combine arbitrary PDB heuristics for MSA with post-hoc optimization. To directly apply other cost partitioning techniques from classical planning, we present a factored representation of the edges in the original search lattice of the MSA problem. We experimentally evaluate, how these techniques compare to existing heuristics in a best-first search environment.

2

Multiple Sequence Alignment

In this chapter, we introduce the formal definition of the Multiple Sequence Alignment problem.

2.1 Formal Definition

Kobayashi and Imai (1998) and Lermen and Reinert (2000) define the MSA problem formally by a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$ that shall be aligned, where each sequence $s_i \in S$ consists of a series of letters of length l_i from an alphabet Σ, e.g. $\Sigma = \{A, C, T, G\}$. Alphabet Σ must not contain the reserved gap letter $\{-\}$. $s_{ij} \in \Sigma$ refers to the j-th letter of sequence s_i.

Definition 2.1.1. *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}$ over alphabet Σ. Let $\Sigma' := \Sigma \cup \{-\}$. An **alignment** of $\mathcal{S}$ is a matrix $\mathcal{A}^{n \times m} = (a_{ij}), 1 \leq i \leq n, 1 \leq j \leq m$, where $a_{ij} \in \Sigma'$ for all $1 \leq i \leq n, 1 \leq j \leq m$ and after deleting all $\{-\}$, a_i is exactly the corresponding sequence s_i and columns with only $\{-\}$ are not allowed.*

The quality of an alignment is evaluated by a score matrix $sub : \Sigma' \times \Sigma' \to \mathbb{N}$, which assigns costs to each pair of letters from alphabet Σ', including the cost for inserting the gap letter. Aligning evolutionary similar letters in the same column leads to fewer costs.

Definition 2.1.2. *Given an alignment $\mathcal{A}$ and score matrix sub. The **pair score** of two sequences s_i, s_j is defined as:*

$$C_{ij}^{\mathcal{A}} = \sum_{k=1}^{m} sub(a_{ik}, a_{jk})$$

*The **sum of pairs score** of alignment $\mathcal{A}$ is defined as:*

$$C^{\mathcal{A}} = \sum_{1 \leq i < j \leq n} C_{ij}^{\mathcal{A}}$$

With this definition, each insertion or extension of a gap is treated and scored equally. This is commonly referred to as *linear gap costs*, where the gap costs are proportional to the gap length. However, one larger mutation of k successive changes in sequence naturally occurs more likely than k single mutations at different positions. To account for this, *affine*

gap costs were introduced by Altschul (1989), where insertion and extension of a gap is distinguished. In practice, gaps are then scored depending on the alignment of the preceding column, where lower costs are assigned to an extension of an existing gap. In the following, we use linear gap costs.

In our experiments, we use the popular PAM250 substitution matrix, introduced by Dayhoff et al. (1978). It scores pairs of amino acids based on the biological likelihood that the first amino acid of that pair mutates to the second during evolution. PAM250 means that 250 mutations per 100 amino acids have been accepted in the genetic code.

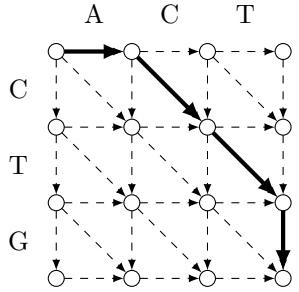
2.2 Shortest Path Problem

Ikeda and Imai (1994) formulate the MSA problem for Sequences $\mathcal{S} = \{s_1, \ldots, s_n\}$ with respective lengths of l_i with $1 \leq i \leq n$ as a shortest path problem through a directed acyclic graph (DAG) $\mathcal{G} = (V, E)$, where

$$V = \{(x_1, \ldots, x_n) \mid x_i = 0, \ldots, l_i\}$$

$$E = \bigcup_{e \in \{0,1\}^n} \{(v, v+e) \mid v, v+e \in V, e \neq 0\}.$$

A score matrix is used to evaluate potential alignments during search in an n-dimensional lattice between two corners. Each of the sequences are considered as one of the n lattice dimensions. Hence, n is called the dimensionality of the MSA problem. A vertex in the lattice is defined by its index, indicating the current position of each sequence. Vertices are connected by edges, such that each sequence can either progress by one or maintain its position. Edges, where all sequences maintain their position are forbidden and omitted. The optimal alignment corresponds to the cheapest path between the two vertices at positions $(0, \ldots, 0)$ and $(l_1, \ldots, l_n)$.



Score matrix:

	A	C	T	G	_
A	0	4	2	2	3
C		1	4	3	3
T			0	6	3
G				1	3
_					0

Alignment $\mathcal{A}$:

A	C	T	_
_	C	T	G
3	+1	+0	+3

$C^{\mathcal{A}} = 7$

Figure 2.1: Two-dimensional optimal sequence alignment using a score matrix.

Figure 2.1 shows an example of aligning two DNA sequences $\langle \mathrm{A}, \mathrm{C}, \mathrm{T} \rangle$ and $\langle \mathrm{C}, \mathrm{T}, \mathrm{G} \rangle$. From left to right, the figure shows the two-dimensional search lattice, the score matrix and the optimal alignment with its cost. In this two-dimensional case, the edge costs are directly given by the score matrix. A horizontal or vertical move corresponds to inserting the gap letter in one of the sequences while progressing the other sequence. A diagonal move progresses both sequences at the same time. The cost of the example alignment is the *pair score* $C^{\mathcal{A}} = sub(\mathrm{A}, \_) + sub(\mathrm{C}, \mathrm{C}) + sub(\mathrm{T}, \mathrm{T}) + sub(\_, \mathrm{G}) = 3 + 1 + 0 + 3 = 7$.

In a case with three or more sequences, edge costs are defined as the sum of the corresponding edges induced by all pairwise projections of the alignment (the sum of pairs score). Finding an alignment with minimal sum of pairs score (Definition 2.1.2) is an optimal solution to the MSA problem.

3
Solving MSA

In this chapter, we first describe the earliest techniques for solving optimal MSA problems. In particular, we describe how to compute optimal alignments with dynamic programming and A* with admissible heuristics. Afterwards, we formally define pattern databases for MSA and show how previous heuristics fall into this category.

3.1 Needleman-Wunsch Algorithm

Needleman and Wunsch (1970) describe a dynamic programming algorithm for computing optimal scores between two biological sequences s_1, s_2 of lengths l_1 and l_2. For a simple definition of the algorithm, we use linear gap costs. The algorithm generates a zero-based index table $T(i, j)$ with $0 \leq i \leq l_1$ and $0 \leq j \leq l_2$ holding optimal scores for each index. First the goal index is initialized to $T(l_1, l_2) = 0$. Then the last row and column are filled with

$$T(i, l_2) = T(i + 1, l_2) + \text{gap cost, for all } l_1 > i \geq 0$$

$$T(l_1, j) = T(l_1, j + 1) + \text{gap cost, for all } l_2 > j \geq 0$$

The rest of the table is then filled backwards using the following formula.

$$T(i, j) = min \begin{cases} T(i + 1, j) + \text{gap cost} \\ T(i, j + 1) + \text{gap cost} \\ T(i + 1, j + 1) + sub(s_{1i}, s_{2j}) \end{cases}$$

Figure 3.1 shows the example from Figure 2.1 solved using the Needleman-Wunsch algorithm. Table entries correspond to vertices in the search graph. The table approach allows to get the optimal alignment as well as the costs for each position to the goal. Obtaining the shortest path can then be done by following the cheapest neighbor from the start to the goal node.

The algorithm can be extended to n-dimensional alignments, but its complexity of $O(l^n)$ for n sequences of lengths l limits its practical application to dimensions of at most three.

	A	C	T	
C	7	4	6	9
T	8	6	3	6
G	8	6	6	3
	9	6	3	0

Score matrix:

	A	C	T	G	_
A	0	4	2	2	3
C		1	4	3	3
T			0	6	3
G				1	3
_					0

Figure 3.1: Needleman-Wunsch score table using a score matrix.

3.2 Heuristic Search

Ikeda and Imai (1994, 1999) use A* (Hart et al., 1968) to solve MSA problems. A* is a best-first search algorithm for graphs that maintains a priority queue of search nodes sorted by their f-values. The f-value of a node is the sum of its g- and h-values, where the g-value is the cost of reaching the vertex (in the search lattice) of the node from the initial vertex, and h is a heuristic estimate of the cost of reaching the goal vertex.

A heuristic h is a function that assigns each vertex v a value $h(v)$ that estimates the cost of reaching the goal vertex from that vertex. A heuristic is perfect, written h^*, if $h(v)$ is the exact cost of reaching the goal vertex from all vertices v. It is admissible if $h(v) \leq h^*(v)$ for all vertices v.

A* is guaranteed to find optimal solutions if the used heuristic is admissible. Hence one central question is how to come up with informed admissible heuristics for the MSA problem.

3.3 Pattern Databases for MSA

Ikeda and Imai (1994) suggest to use the sum of the costs of all pairwise optimal alignments as admissible heuristic estimate for higher-dimensional alignments. We generalize this idea and define **pattern databases** for MSA.

Definition 3.3.1. *Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a family of sequences. An (ordered) subset $P \subseteq \mathcal{S}$ with $|P| \geq 2$ is called a **pattern**. The **pattern database (PDB)** heuristic, written h^P equals the perfect heuristic h^* for the subproblem induced by the pattern P.*

PDBs are called databases because the perfect heuristic for a subproblem is usually computed once and stored in a simple look-up table that allows fast acces of heuristic values during search. To access the corresponding PDB value of a vertex v on the search graph induced by $\mathcal{S}$, we project v to its corresponding vertex on the search graph induced by P by removing all indices from v of sequences that do not appear in P. In the following, we refer to the search graph induced by $\mathcal{S}$ as search graph if $\mathcal{S}$ refers to the original problem. With this definition of PDBs, we can now formalize the **pairwise heuristic** of Ikeda and Imai (1994).

Definition 3.3.2 (Ikeda and Imai, 1994)**.** *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$, let h^{ij} be the PDB for pattern $\{s_i, s_j\}$. The **pairwise heuristic** for a vertex v in the*

search graph is

$$h_{pair}(v) = \sum_{1 \leq i < j \leq n} h^{ij}(v)$$

They compute and store all h^{ij} using the algorithm by Needleman and Wunsch (1970). They further show that h_{pair} is admissible.

More generally, the tables computed with the Needleman and Wunsch (1970) algorithm can be seen as the look-up table used in two-dimensional PDBs because they store perfect heuristic values for all abstract search vertices of the subproblem. Alternatively to using dynamic programming, one could also use an exhaustive search version of Dijkstra's algorithm (Dijkstra, 1959) to compute PDBs.

Kobayashi and Imai (1998) generalize the h_{pair} heuristic by extending it from using only the information of two-dimensional subproblems to k-fold subproblems of sizes $k \geq 3$. We formalize their $\boldsymbol{h_{all\text{-}k}}$ heuristic again as combination of PDBs.

Definition 3.3.3 (Kobayashi and Imai, 1998)**.** *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}$, $n \geq 2$, let $h^{x_1,\ldots,x_k}$ be a PDB of size k for pattern $\{s_{x_1}, \ldots, s_{x_k}\}$. The $\boldsymbol{h_{all\text{-}k}}$ heuristic for a vertex v in the search graph is*

$$h_{all\text{-}k}(v) = \frac{1}{\binom{n-2}{k-2}} \sum_{1 \leq x_1 < \cdots < x_k \leq n} h^{x_1,\ldots,x_k}(v)$$

This heuristic combines the information of all k-fold subalignments. When chosing all k-fold patterns out of n sequences, every sequence pair appears $\binom{n-2}{k-2}$ times in the patterns. Because we defined the cost function as the sum of pairs score, the overall estimate needs to be normalized by the number of sequences appearing pairwise in the patterns in order to ensure admissibility. They also prove that $h_{all\text{-}k}$ gives a tighter lower bound on the perfect heuristic h^* than h_{pair}, i.e. $h_{pair}(v) \leq h_{all\text{-}k}(v) \leq h^*(v)$ *for all* $v \in V$. The $h_{all,2}$ heuristic is equal to the h_{pair} heuristic.

They define another heuristic $\boldsymbol{h_{one\text{-}k}}$ that uses less higher-dimensional PDBs in order to fit the heuristic in memory. They use one k-fold, one $(n-k)$-fold and 2-fold alignments.

Definition 3.3.4 (Kobayashi and Imai, 1998)**.** *Given a family of sequences $S = \{s_1, \ldots, s_n\}$, $n \geq 3$, let $h^{x_1,\ldots,x_k}$ be the PDB of an arbitrary chosen pattern of size k, and $h^{x_{k+1},\ldots,x_n}$ be the PDB of the remaining $(n-k)$ sequences not appearing in the k-fold pattern. The $\boldsymbol{h_{one\text{-}k}}$ heuristic for a vertex v in the search graph is*

$$h_{one\text{-}k}(v) = h^{x_1,\ldots,x_k}(v) + h^{x_{k+1},\ldots,x_n}(v) + \sum_{i=1}^{k} \sum_{j=k+1}^{n} h^{x_i,x_j}(v)$$

They further show that $h_{pair}(v) \leq h_{one\text{-}k}(v) \leq h^*(v)$ *for all* $v \in V$. This makes sense, because the cost of the sum of all m-fold optimal alignments is dominated by the cost of a k-fold optimal alignment for $m \leq k$ when choosing the same sequences (Carrillo and Lipman, 1988). Choosing numbers of k close to $\frac{1}{2}n$ intuitively works best, because this minimizes the space and time used to store and calculate the k-fold and $(n-k)$-fold heuristics. Also note, that the overall estimate does not need to be normalized due to the fact, that the higher-dimensional patterns are chosen to be disjunct and the 2-fold patterns consist of one sequence of each of the two higher-dimensional patterns.

4

Combining Multiple Pattern Databases

With multiple admissible heuristics available, it would be ideal to combine all heuristics by summing their estimates. Unfortunately, the sum of heuristics is not guaranteed to be admissible, especially if the heuristics consider overlapping parts of the problem. The trivial solution to admissibly combine these heuristics is to use the maximum heuristic value in each state. This selects the single most accurate estimator among the heuristics while staying admissible. However, this approach yields a loss of all the information that other heuristics could have contributed to the estimate.

The motivation of **Cost Partitioning (CP)** is to get better estimates than using the maximum estimate of the available heuristics. The idea is to combine multiple heuristic values by distributing operator costs between the heuristics in order to stay admissible. Since each pattern only considers parts of the problem, more aspects of the problem can be taken into account by combining their estimates.

In the following, we introduce non-conflicting pattern collections for MSA and show how they can be combined to create additive heuristics for MSA. We introduce a factored representation of operators for the MSA search lattice. Then we apply different cost partitioning techniques to the MSA problem.

4.1 Additive Heuristics

To discuss how to sum heuristics without violating admissibility, we first define the notion of additive heuristics.

Definition 4.1.1. *A set of heuristics $\mathcal{H} = \langle h_1, \ldots, h_n \rangle$ for an MSA problem $\mathcal{S}$ over Σ is **additive** if their sum is admissible, i.e., if $\sum_{i=1}^{n} h_i(v) \leq h^*(v)$ for all vertices v of the search graph induced by $\mathcal{S}$.*

In this work, we are concerned with sets of PDB. To reason about their additivity, we need to define properties for their underlying patterns.

Definition 4.1.2. *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$. A **pattern collection** of S is a collection $\mathcal{P} = \{P_1, \ldots, P_m\}$ where $P_i \subseteq S$ for all $1 \leq i \leq m$. A pair P_i, P_j of*

*patterns in $\mathcal{P}$ **conflict** if $|P_i \cap P_j| > 1$. $\mathcal{P}$ is called **non-conflicting** if no pair of elements of $\mathcal{P}$ conflict, otherwise it is called conflicting.*

Non-conflicting pattern collections ensure that no pair of sequences is contained in more than one pattern. It is easy to see that non-conflicting pattern collections are additive because costs are always defined for pairs of sequences, and no pair of sequences is contained in more than one pattern.

Theorem 1. *The pattern collection heuristic $h^{\mathcal{P}}$ is **admissible** when given a non-conflicting pattern collection.*

Proof. Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}$ and any alignment $\mathcal{A}$ for $\mathcal{S}$ and let $C_{ij}^{\mathcal{A}}$ be the cost function for 2 sequences s_i, s_j. Given a pattern collection $\mathcal{P} = \{P_1, \ldots, P_m\}$ for $\mathcal{S}$ and let P_{ki} be the i-th sequence of pattern k. If $\mathcal{P}$ is non-conflicting, written $|P_i \cap P_j| < 2$ for all $1 \leq i < j \leq m$, then admissibility follows from the requirement:

$$\sum_{1 \leq i < j \leq n} C_{ij}^{\mathcal{A}} \geq \sum_{k=1}^{m} \sum_{1 \leq i < j \leq |P_k|} C_{P_{ki} P_{kj}}^{\mathcal{A}}$$

□

With non-conflicting pattern collections we can now define a more general version of the heuristics from Kobayashi and Imai (1998) that is admissible for arbitrary non-conflicting pattern collections.

Definition 4.1.3. *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$ and a pattern collection $\mathcal{P} = \{P_1, \ldots, P_m\}$. Let h^{P_i} be the PDB for pattern P_i. The **pattern collection heuristic** $h^{\mathcal{P}}$ for a position v in the search graph is*

$$h^{\mathcal{P}}(v) = \sum_{i=1}^{m} h^{P_i}(v)$$

The pattern collection heuristic equals h_{pair}, if $\mathcal{P}$ contains exactly all possible pairwise patterns of $\mathcal{S}$.

4.2 Canonical PDB Heuristic

If a pattern collection is conflicting, summing PDB values may violate admissibility. However, such a collection may still exhibit parts that are non-conflicting. The canonical PDB heuristic (Haslum et al., 2007) exploits such non-conflicting subsets optimally. To do so, it computes the maximal non-conflicting subsets of the collection and adds heuristic values within each subset and maximizes over all resulting summed heuristic values.

Definition 4.2.1. *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$ and a pattern collection $\mathcal{P}$. Let MNS be the maximal (w.r.t. set inclusion) non-conflicting subsets of $\mathcal{P}$. The **canonical PDB heuristic** for a vertex v of the search graph is*

$$h^{CAN}(v) = \max_{S \in MNS} \sum_{P \in S} h^{P}(v).$$

4.3 Post-Hoc Optimization

Pommerening et al. (2013) show that even better heuristic values can be derived from conflicting pattern collections by applying the so-called **post-hoc optimization** technique. The idea is to constrain the component heuristics based on their conflicts. Then linear programming is used to solve the constrained problem.

In order to identify the constraints of the components we define **strictly conflicting** pattern collections.

Definition 4.3.1. *A pattern collection $\mathcal{P} = \{P_1, \ldots, P_m\}$ is called **strictly conflicting** if $|\bigcap_{i=0}^{m} P_i| > 1$.*

This means that all elements of the pattern collection conflict pairwise on the same pair of elements. Strictly conflicting pattern collections can be seen as the complement to maximal non-conflicting subsets of a pattern collection used in the canonical PDB heuristic (Definition 4.2.1). Pommerening et al. (2013) show that the dual of the linear program solved for the post-hoc optimization heuristic equals the canonical heuristic value when constraining the LP to integer solutions. With the notion of strictly conflicting pattern collections we can now define the h_{PHO} heuristic for MSA.

Definition 4.3.2. *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}$ and a pattern collection $\mathcal{P} = \{P_1, \ldots, P_m\}$. Let h^{P_i} be the PDB of pattern $P_i \in \mathcal{P}$. Let $\langle \omega_1, \ldots, \omega_m \rangle$ be a solution to the linear program that maximizes the **post-hoc optimization** heuristic $h^{PHO}(v)$ for vertex v in the search graph:*

$$h^{PHO}(v) = \sum_{i=1}^{m} \omega_i \cdot h^{P_i}(v)$$

$$s.t. \sum_{i:P_i \in S'} \omega_i \leq 1 \text{ for all strictly conflicting pattern collections } S' \subseteq \mathcal{P}$$

$$s.t.\ 0 \leq \omega_i \leq 1 \text{ for all } P_i$$

For each pair of sequences that appears in multiple patterns, we need to make sure that the corresponding weights of the affected patterns add up to at most one. To get all strictly conflicting pattern collections in practice, we create a bucket for each possible pair of sequences. We then store identifiers of patterns that contain a certain pair of sequences in their corresponding bucket. Then we use IBM ILOG CPLEX[1] to create the constraints and for solving the linear program. We need to solve the linear program for every vertex in the search graph we encounter during search. Comparing to the heuristics considered so far, this leads to a lot of time being spent in the search to compute the heuristic value.

As we later confirm in our experiments, the estimate of $h_{all,3}$ is at least as good as the estimate of h^{PHO}. Only if the pattern collection used for post-hoc optimization is chosen to consist of exactly all possible 3-fold alignments, its outcome is the same as $h_{all,k}$. Nonetheless, even in this case h^{PHO} will never give better estimates than $h_{all,k}$. The reason lies in the interdependencies of the patterns.

[1] IBM ILOG CPLEX Optimization Studio Version 12.8.0, http://www-01.ibm.com/support/docview.wss?uid=swg24044295

Theorem 2. *$h_{all,k}$ dominates h^{PHO}.*

Proof sketch. Given four sequences of length one $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$ and a pattern collection $\mathcal{P} = \{P_1 = \{s_1, s_2, s_3\}, P_2 = \{s_1, s_2, s_4\}, P_3 = \{s_1, s_3, s_4\}, P_4 = \{s_2, s_3, s_4\}\}$. The requirement for h^{PHO} to give a better estimate than $h_{all,k}$ is, that the estimate of one pattern is significantly higher than all others. We try to achieve this for P_1 by chosing a score matrix, where the costs of aligning any sequence with s_4 has the minimal cost of 0 and aligning all other pairs or adding a gap costs 1. This results in the following intial heuristic estimates for start vertex s: $h^{P_1}(s) = 3$, $h^{P_2}(s) = 1$, $h^{P_3}(s) = 1$, $h^{P_4}(s) = 1$. The linear program maximizes the estimate to $h^{PHO}(s) = 1 \cdot 3 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 = 3 = \frac{3+1+1+1}{2} = h_{all,3}(s)$. □

4.4 A Factored Representation of MSA with Operators

To directly apply cost partitioning techniques from planning, we present a factored representation of MSA with operators in the following. Consider a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$. In this representation, each edge of the search graph can be factored as a set of operators. In particular, for each pair of sequences $s_i, s_j \in \mathcal{S}$, we define operators $o^{i,j}_{\langle x,y\rangle \to \langle x',y'\rangle}$ for each edge $\langle x, y\rangle \to \langle x', y'\rangle$ of the search graph induced by $\{s_i, s_j\}$.

This set of operators for $\mathcal{S}$, written $\mathcal{O}$, contains the basic factors that we can use to represent edges in search graphs of higher dimensions, i.e., over at least three sequences. Each edge in such a search graph can be represented as a combination of operators from the pairs over all sequences. In particular, each pair of sequences contributes exactly one operator if the edge in the search graph changes at least one of the sequences of that pair, and no operator otherwise.

Given n sequences, we formalize all **factored operators** as $\mathcal{O} = \{o^{i,j}_{\langle x,y\rangle \to \langle x',y'\rangle} \mid 1 \leq i < j \leq n, 0 \leq x \leq l_i, 0 \leq y \leq l_j\}$.

An operator $o^{i,j}_{\langle x,y\rangle \to \langle x',y'\rangle}$ *affects* heuristic h^P if the pattern P contains the sequences s_i, s_j. We write $\mathit{aff}(h^P) = \{o \in \mathcal{O} \mid o \text{ affects } h^P\}$.

As an example, consider the edge $\langle 3, 3, 5\rangle \to \langle 4, 3, 6\rangle$ of a three-dimensional search graph. This edge has the factored representation consisting of the following 3 operators: $\{o^{1,2}_{\langle 3,3\rangle \to \langle 4,3\rangle}, o^{1,3}_{\langle 3,5\rangle \to \langle 4,6\rangle}, o^{2,3}_{\langle 3,5\rangle \to \langle 3,6\rangle}\}$.

One advantage of this factored representation is that we can now associate a path-independent cost with each operator. The cost of operator $c(o^{i,j}_{\langle x,y\rangle \to \langle x',y'\rangle})$ is defined by the entry of the score matrix at index $sub(s_{ix'}, s_{jy'})$ because applying the operator means to align exactly the nucleobase pair at position x', y'.

While this formalization requires a large number of operators, it is still much smaller than defining a single operator for each edge in the original search graph, because intuitively speaking, the number of operators is effectively reduced from all possible edges in the lattice to all edges on its 2-faces.

With this formalization, we can now compute PDB heuristics using specific cost functions for operators that may result in different values than using the original score matrix.

Definition 4.4.1. *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$ and let $P \subseteq \mathcal{S}$ be a pattern for $\mathcal{S}$. Let further $\mathcal{O}$ be the set of operators for $\mathcal{S}$ as above. Let c be a an arbitrary*

cost function for $\mathcal{O}$. The PDB heuristic $h^{P,c}$ is the perfect heuristic for the subproblem induced by P, however using cost function c for computing entries in the PDB.

4.5 Cost Partitioning

We are now ready to adapt cost partitioning techniques from classical planning to MSA problems, using the notion of operators and operator cost functions. The following is based on notation by Seipp et al. (2017).

Definition 4.5.1. *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$ and let $\mathcal{P} = \langle P_1, \ldots, P_m \rangle$ be a pattern collection of $\mathcal{S}$. Let $\mathcal{O}$ be the operators for $\mathcal{S}$ and c be the cost function induced by the score matrix.*

*A **cost partitioning** over $\mathcal{P}$ is a tuple $\mathcal{C} = \langle c_1, \ldots, c_n \rangle$ of cost functions whose sum is bounded by $c : \sum_{i=1}^{n} c_i(o) \leq c(o)$ for all $o \in \mathcal{O}$. The cost-partitioned heuristic $h^{\mathcal{C}}$ is defined as $h^{\mathcal{C}}(v) := \sum_{i=1}^{n} h^{P_i, c_i}(v)$ for all vertices v of the search graph over $\mathcal{S}$.*

While we did not need the factored operator representation to define post-hoc optimization, it still can be seen as a form of cost partitioning over the original PDBs, because it partitions costs among them such that every pair of sequences occuring in the PDBs receives a combined weight of at most one.

4.6 Greedy Zero-One Cost Partitioning

Greedy zero-one cost partitioning (Haslum et al., 2005; Edelkamp, 2006) assigns the full cost of each operator to at most one of the available PDBs affected by that operator. When multiple PDBs are affected by an operator it greedily choses the first PDB by a predefined ordering to assign the full costs to.

Definition 4.6.1 (Seipp et al., 2017)**.** *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$ and let $\mathcal{P} = \langle P_1, \ldots, P_m \rangle$ be a pattern collection of $\mathcal{S}$ and let $\Omega(\mathcal{P})$ denote the set of orders of $\mathcal{P}$ consisting of all of its permutations. Let h^{P_i} be the PDB of pattern $P_i \in \mathcal{P}$. For a given order $\omega = \langle h^{P_1}, \ldots, h^{P_m} \rangle \in \Omega(\mathcal{P})$, the **greedy zero-one cost partitioning** is the tuple $\mathcal{C} = \langle c_1, \ldots, c_m \rangle$ where*

$$c_i(o) = \begin{cases} c(o) & \text{if } o \in \mathit{aff}(h^{P_i}) \text{ and } o \notin \cup_{j=1}^{i-1} \mathit{aff}(h^{P_j}) \\ 0 & \text{otherwise} \end{cases}$$

for all $o \in \mathcal{O}$. We write h_{ω}^{GZOCP} for the heuristic that is cost-partitioned by greedy zero-one cost partitioning for order ω.

Unlike post-hoc optimization, we do not need to recompute the heuristic value in each state when using zero-one cost partitioning. Instead, the heuristic is precomputed once before search and then applied to every state.

For every pair of sequences appearing in the patterns, we store all operators in a two-dimensional table. We then create the PDBs in a given order by using these operators.

After creating each PDB, all operators used in this PDB are consumed and set to 0. The next PDB using the operators of a pair of sequences already consumed can then be added admissibly to the overall estimate.

4.7 Saturated Cost Partitioning

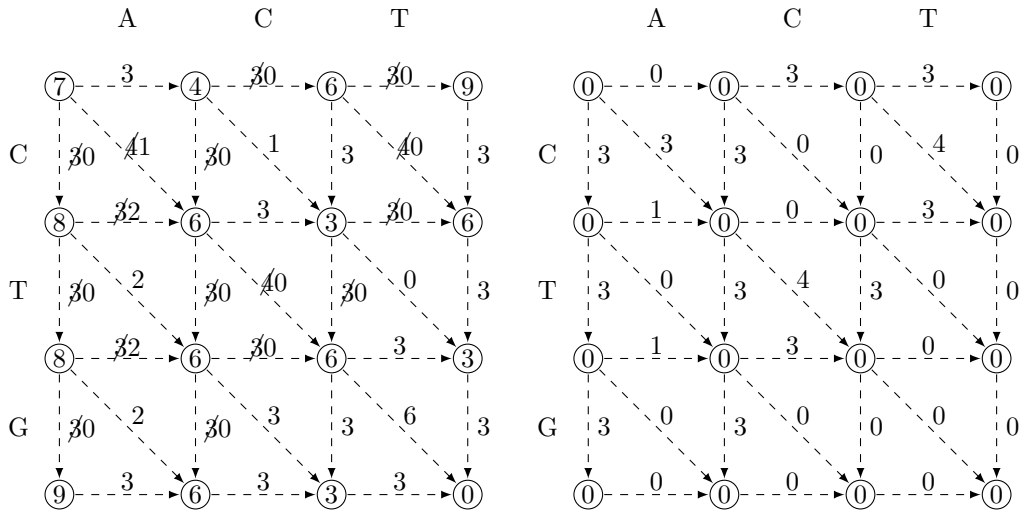
Seipp and Helmert (2014) proposed saturated cost partitioning to overcome some of the shortcomings from greedy zero-one cost partitioning. In particular, instead of assigning the full costs to a single operator it can also assign parts of the full costs to one component until it is *saturated* while the remainder can contribute to other components. Like greedy zero-one cost partitioning, saturated cost partitioning choses an order in which the components are considered.

The **saturated cost function** for the heuristic h^P and cost c, written $\mathit{saturate}(h^P, c)$ is the minimal cost function $c' \leq c$ with $h^{c'}(v) = h^c(v)$ for all vertices v in the search graph.

Definition 4.7.1 (Seipp et al., 2017)**.** *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$ and let $\mathcal{P} = \langle P_1, \ldots, P_m \rangle$ be a pattern collection of $\mathcal{S}$. Given an order $\omega = \langle h^{P_1}, \ldots, h^{P_m} \rangle \in \Omega(\mathcal{P})$, the* ***saturated cost partitioning*** *$\mathcal{C} = \langle c_1, \ldots, c_n \rangle$ and the remaining cost functions $\langle \bar{c}_0, \ldots, \bar{c}_n \rangle$ are defined by*

$$\begin{aligned} \bar{c}_o &= c \\ c_i &= \mathit{saturate}(h^{P_i}, \bar{c}_{i-1}) \\ \bar{c}_i &= \bar{c}_{i-1} - c_i \end{aligned}$$

We write h^{SCP}_ω for the heuristic that is cost-partitioned by saturated cost partitioning for order ω.



(a) Saturated operator costs (b) Remaining operator costs

Figure 4.1: Saturated and remaining operator costs for a two-dimensional PDB.

We show an example of how the saturated and remaining costs for a two-dimensional PDB look like in Figure 4.1. After computing the optimal heuristic values for each state of the PDB with the score matrix shown in Figure 3.1, each operator is assigned the new saturated cost. The saturated cost for each operator is the least possible cost greater than 0 without changing the heuristic outcome of the PDB. These costs can not be exploited by this PDB and can be assigned to the next PDB affected by the same operators.

4.8 Uniform Cost Partitioning

Katz and Domshlak (2008) proposed uniform cost partitioning. Unlike greedy zero-one and saturated cost partitioning, no ordering is needed to compute the heuristic, because each affected component is assigned an equal part of the operator cost.

Definition 4.8.1 (Seipp et al., 2017)**.** *Given a family of sequences $\mathcal{S} = \{s_1, \ldots, s_n\}, n \geq 2$ and let $\mathcal{P} = \langle P_1, \ldots, P_m \rangle$ be a pattern collection of $\mathcal{S}$. The **uniform cost partitioning** is the tuple $\mathcal{C} = \langle c_1, \ldots c_n \rangle$, where for all $o \in \mathcal{O}$*

$$c_i(o) = \begin{cases} \frac{c(o)}{|P_i \in \mathcal{P} | o \in \mathit{aff}(h^{P_i})|} & \text{if } o \in \mathit{aff}(h^{P_i}) \\ 0 & \text{otherwise.} \end{cases}$$

We write h^{UCP} for the heuristic that is cost-partitioned by uniform cost partitioning.

The $h_{all,k}$ heuristic (Definition 3.3.3) also uses a simple form of uniform cost partitioning, where the cost functions of the PDBs remain unmodified. The difference is, that $h_{all,k}$ uses PDBs instead of operators as the smallest components to partition the costs among.

5 Experiments

In this chapter, we evaluate the estimates of some of the heuristics from chapter 4 and compare them to the heuristics shown in chapter 3 (Ikeda and Imai, 1994; Kobayashi and Imai, 1998).

5.1 Experimental Setup

For all our experiments, we use the MSASolver[2] program by Matthew Hatem as a basis. MSASolver is an in-memory best-first search solver for MSA written in Java. Included is the BAliBASE Benchmark Reference Set 1 (Thompson et al., 1999) and the PAM250 (Dayhoff et al., 1978) score matrix. The 82 instances in the benchmark set range from 3 to 6 sequences with sequence lengths between 58 and 993. It implements 2-fold and 3-fold sub-alignments computed with dynamic programming using affine gap costs and the h_{pair} and $h_{one,3}$ heuristics, that combines these sub-alignments in their heuristic estimates.

We implemented the $h_{all,3}$, h^{PHO} and h_w^{GZOCP} heuristics for MSASolver. For simplicity and memory reasons, we modified the sub-alignments to use linear gap costs. This reduces the required memory to store a 3-fold alignment by a factor of $2^3 - 1$, because we do not need to store the information, whether a gap is already existing for each parent state. This allows us to store the $h_{all,3}$ heuristic for 73 of the 75 easiest instances.

All experiments were run on an Intel(R) Core(TM) i5-6600K CPU @ 3.5GHz machine with 16GB of main memory.

5.2 Choosing Patterns

The estimate of our implemented heuristics depend heavily on the patterns we choose. We can intuitively see, that the information of all pairs of sequences should be used to maximize the heuristic outcome. Because of memory limitations, the highest dimension of sub-alignments we use is three.

The existing heuristics from chapter 3 essentially use certain patterns. In particular,

[2] https://github.com/matthatem/MSASolver

h_{pair} uses all possible 2-fold patterns, $h_{one,3}$ uses one or two (depending on the number of sequences) non-conflicting 3-fold patterns and all remaining 2-fold patterns and $h_{all,3}$ uses all possible 3-fold patterns. We use the same patterns that each of the existing heuristics uses and compare them to our implemented heuristics. Additionally, we evaluate how our heuristics perform on other conflicting pattern collections than the pattern collection used for $h_{all,3}$.

5.3 Results

Table 5.1 shows the initial heuristic values for instance 1aab_ref1 for each implemented heuristic given the pattern collections the existing heuristics implicitly use. Results for all instances can be found in Appendix A.

1aab_ref1.seq (4 sequences)					
Pattern collection in order ω	$h_{pair}(s)$	$h_{one,3}(s)$	$h_{all,3}(s)$	$h^{PHO}(s)$	$h_{\omega}^{GZOCP}(s)$
$\{0,1\},\{0,2\},\{0,3\},\{1,2\},\{1,3\},\{2,3\}$	14179	-	-	14179	14179
$\{0,1,2\},\{0,3\},\{1,3\},\{2,3\}$	-	14199	-	14199	14199
$\{0,1,2\},\{0,1,3\},\{0,2,3\},\{1,2,3\}$	-	-	14302	14302	14199

Table 5.1: Initial heuristic estimate comparison for the instance 1aab_ref1.seq

5.3.1 Greedy Zero-One Cost Partitioning

We find that greedy zero-one cost partitioning can only exploit higher-dimensional patterns, if none of their operators have already been consumed by an earlier component in the order ω. When we choose all possible 3-fold alignments for our pattern collection, all operators $o^{0,1}$, $o^{0,2}$ and $o^{1,2}$ are consumed by the first PDB induced by $\{0,1,2\}$. The second again uses $o^{0,1}$ resulting in its remaining estimate being the same as it would be using the sum of the PDBs induced by $\{0,2\}$ and $\{1,2\}$. The third pattern only contributes the estimate of the PDB induced by $\{2,3\}$ and the last pattern does not contribute anything, because all operators have already been used. In this case, h_{w}^{GZOCP} gives the same estimate as using just one 3-fold alignment and the remaining 2-fold alignments. In the latter case, the best heuristic estimate h_{w}^{GZOCP} can achieve is the same as $h_{one,3}$ depending on the order ω. The same behaviour can be observed, when a 2-fold pattern is used before a conflicting 3-fold PDB. In this case, the estimate is the same as it would be just using all 2-fold PDBs.

5.3.2 Post-Hoc Optimization

Post-hoc optimization gives the best possible combination and cost partitioning among the PDBs for a given abstraction. When no pattern conflicts, the linear program does assign weights of one for every component resulting in the same estimate that h_{pair} generates. When using only one 3-fold PDB, the best possible outcome is the same estimate that

$h_{one,3}$ generates. All 2-fold PDBs that conflict with the 3-fold PDB are assigned weights of zero and do not contribute to the estimate. When we choose all possible 3-fold alignments as our pattern collection the outcome is the same as $h_{all,3}$. We show different heuristic outcomes when using different pattern collections in Table 5.2.

We noticed, that when using up to two 3-fold PDBs, the linear program optimizes the weights to either 1 or 0. When using three or more 3-fold PDBs, it weights every PDB by either 0.5 or 0. We think this might be an implicit property of the given optimization problem.

1aab_ref1.seq (4 sequences)		
#3-fold	Pattern collection	$h^{PHO}(s)$
1	$\{0,1,2\},\{0,3\}$	9466
	$\{0,1,2\},\cancel{\{0,1\}},\cancel{\{0,2\}},\cancel{\{1,2\}},\{0,3\},\{1,3\},\{2,3\}$	14199
	$\{0,1,2\},\{0,3\},\{1,3\},\{2,3\}$	14199
2	$\{0,1,2\},\cancel{\{0,1,3\}},\cancel{\{0,1\}}$	7186
	$\{0,1,2\},\cancel{\{0,1,3\}},\{2,3\}$	9466
	$\cancel{\{0,1,2\}},\{0,1,3\},\{1,2\},\cancel{\{1,3\}},\{2,3\}$	11849
	$\{0,1,2\},\cancel{\{0,1,3\}},\{0,3\},\{1,3\},\{2,3\}$	14199
3	$\{0,1,2\},\{0,1,3\},\{0,2,3\},\{1,2\}$	11951
	$\{0,1,2\},\{0,1,3\},\{0,2,3\},\{1,2\},\{1,3\}$	13119
	$\{0,1,2\},\{0,1,3\},\{0,2,3\},\cancel{\{0,1\}},\cancel{\{0,2\}},\cancel{\{0,3\}},\{1,2\},\{1,3\},\{2,3\}$	14259
	$\{0,1,2\},\{0,1,3\},\{0,2,3\},\{1,2\},\{1,3\},\{2,3\}$	14259
4	$\{0,1,2\},\{0,1,3\},\{0,2,3\},\{1,2,3\}$	14302

Table 5.2: Initial heuristic estimates of h^{PHO} for the instance 1aab_ref1.seq. Crossed out patterns are assigned weights of 0 by the linear program

6 Conclusion

In this master thesis, we looked into the Multiple Sequence Alignment (MSA) problem. We showed how existing heuristics can be interpreted as pattern collection heuristics, and how arbitrary pattern collections for MSA can be used in cost partitioning heuristics. We implemented post-hoc optimization and greedy zero-one cost partitioning for MSA and evaluated how they perform compared to the existing heuristics. We introduced a factored representation for the operators in the MSA search lattice such that cost partitioning techniques from classical planning can be applied to MSA.

We conclude, that h_w^{GZOCP} has no benefit to the existing heuristics. While its advantage over the existing heuristics is, that we can use arbitrary pattern collections and do not need to identify conflicting patterns, it never performs better than the existing $h_{one,k}$ heuristic. Also, identifying the conflicts can be done in a precomputation step with minimal computational effort and then be used in other heuristics, like h^{PHO}.

While the post-hoc optimization heuristic performs better in most cases, the additional heuristic information comes with some backdraws. Firstly, the heuristic requires the linear program to recalculate the optimal component weights in every search step leading to a large overhead in the main search. Secondly, sometimes choosing pattern collections with more 3-fold PDBs lead to the expensively computed PDB not being considered at all, because the linear program decides, it can not contribute to the estimate based on the constraints.

6.1 Future Work

It would be interesting to implement and evaluate other cost partitioning techniques for MSA. Because choosing a good pattern collection by hand seems not to be easy for post-hoc optimization, one could generate and choose PDBs automatically (Haslum et al., 2007). The merge-and-shrink heuristic (Dräger et al., 2009; Helmert et al., 2014) is another abstraction heuristic, that could make use of the factored operator representation. It would be interesting to apply it to the MSA problem.

Bibliography

Stephen F Altschul. Gap costs for multiple sequence alignment. *Journal of theoretical biology*, 138:297–309, 1989.

Humberto Carrillo and David Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, 48:1073–1082, 1988.

MO Dayhoff, RM Schwartz, and BC Orcutt. A model of evolutionary change in proteins. In *Atlas of protein sequence and structure*, volume 5, pages 345–352. National Biomedical Research Foundation Silver Spring, MD, 1978.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1:269–271, 1959.

Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer*, 11(1):27–37, 2009.

Stefan Edelkamp. Automated creation of pattern database search heuristics. In *International Workshop on Model Checking and Artificial Intelligence*, pages 35–50. Springer, 2006.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Patrik Haslum, Blai Bonet, Héctor Geffner, et al. New admissible heuristics for domain-independent planning. In *AAAI*, volume 5, pages 9–13, 2005.

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, Sven Koenig, et al. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, pages 1007–1012, 2007.

Matthew Hatem and Wheeler Ruml. External memory best-first search for multiple sequence alignment. In *AAAI*, 2013.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3):16, 2014.

Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89:10915–10919, 1992.

Takahiro Ikeda and Hiroshi Imai. Fast a algorithms for multiple sequence alignment. *Genome Informatics*, 5:90–99, 1994.

Takahiro Ikeda and Hiroshi Imai. Enhanced a* algorithms for multiple alignments: optimal alignments for several sequences and k-opt approximate alignments for large cases. *Theoretical Computer Science*, 210:341–374, 1999.

Michael Katz and Carmel Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In *ICAPS*, pages 174–181, 2008.

Hirotada Kobayashi and Hiroshi Imai. Improvement of the a* algorithm for multiple sequence alignment. *Genome Informatics*, 9:120–130, 1998.

Martin Lermen and Knut Reinert. The practical use of the a* algorithm for exact multiple sequence alignment. *Journal of Computational Biology*, 7:655–671, 2000.

Saul B Needleman and Christian D Wunsch. 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, 1970.

Florian Pommerening, Gabriele Röger, and Malte Helmert. Getting the most out of pattern databases for classical planning. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 2357–2364, 2013.

Stefan Schroedl. An improved search algorithm for optimal multiple-sequence alignment. *Journal of Artificial Intelligence Research*, 23:587–623, 2005.

Jendrik Seipp and Malte Helmert. Diverse and additive cartesian abstraction heuristics. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014). AAAI Press*, pages 289–297, 2014.

Jendrik Seipp, Thomas Keller, and Malte Helmert. A comparison of cost partitioning algorithms for optimal classical planning. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017). AAAI Press*, 2017.

Julie D. Thompson, Frédéric Plewniak, and Olivier Poch. Balibase: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics (Oxford, England)*, 15:87–88, 1999.

Iain M Wallace, Orla O'sullivan, Desmond G Higgins, and Cedric Notredame. M-coffee: combining multiple sequence alignment methods with t-coffee. *Nucleic acids research*, 34: 1692–1699, 2006.

Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1:337–348, 1994.

Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. A* with partial expansion for large branching factor problems. In *AAAI/IAAI*, pages 923–929, 2000.

A

Appendix

Instance	$h_{pair}(s)$	$h_{one,3}(s)$	$h_{all,3}(s)$
1aab	14179	14199	14302
1aboA	22380	22496	22664
1ac5	86117	86447	86748
1ad2	39202	39268	39320
1ad3	78476	78542	78663
1adj	75774	75822	75950
1aho	19202	19260	19320
1ajsA	75432	75930	76287
1amk	73200	73246	73332
1ar5A	35262	35312	35351
1aym3	43912	43980	44155
1bbt3	61648	61792	62343
1bgl	180887	181241	-
1cpt	80050	80522	80846
1csp	19740	19740	19762
1csy	32208	32250	32358
1dlc	109189	109443	109721
1dox	17510	17528	17565
1eft	73291	73391	73603
1ezm	87452	87462	87608
1fieA	122542	122700	122866
1fjlA	31976	32010	32112
1fkj	31454	31484	31584
1fmb	17492	17492	17513
1gdoA	47842	48066	48236
1gowA	89625	89971	90301
1gpb	235778	235928	-

1gtr	127932	128024	128304
1havA	62804	63020	63550
1hfh	38172	38208	38329
1hpi	13685	13737	13766
1idy	18706	18746	18822
1krn	22980	22980	23021
1ldg	59604	59704	59812
1led	43397	43459	43608
1lvl	86616	86956	87493
1mrj	48099	48187	48266
1ped	35472	35950	35950
1pfc	34946	34980	35084
1pgtA	40054	40164	40288
1pii	48724	48786	48990
1pkm	83513	83689	83844
1plc	28218	28232	28312
1ppn	63216	63270	63339
1pysA	45391	45471	45571
1r69	14315	14393	14438
1rthA	159296	159336	159624
1sbp	84596	84840	85506
1sesA	133794	133966	134291
1tgxA	11147	11165	11259
1thm	49975	49991	50058
1tis	83460	83560	83739
1ton	74946	75100	75450
1tvxA	13021	13139	13211
1ubi	17125	17221	17287
1uky	41520	41792	42030
1wit	32572	32626	32788
1ycc	21054	21092	21240
1zin	38324	38382	38413
2cba	79214	79382	79818
2fxb	16354	16358	16362
2hsdA	50115	50335	50604
2mhr	33280	33292	33346
2pia	53071	53347	53608
2trx	18404	18456	18632
3cyr	19544	19614	19666
3grs	46819	46991	47252
3pmg	98190	98342	98449

451c	26278	26302	26511
4enl	38084	38562	38562
5ptp	69548	69630	69721
9rnt	29072	29078	29105
actin	113450	113508	113666
glg	147708	147930	148536
kinase	90060	90352	90985

Table A.1: Initial heuristic estimates for the BAliBASE Reference Set 1

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Mirko Riesterer

Matriculation number — Matrikelnummer
11-064-011

Title of work — Titel der Arbeit
Cost Partitioning Techniques For Multiple Sequence Alignment

Type of work — Typ der Arbeit
Master thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 21.08.2018

M. Riesterer

Signature — Unterschrift