GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

ISSN 1612-6793

Bachelor's Thesis

submitted in partial fulfillment of the
requirements for the course "Applied Computer Science"

Multiple sequence alignment based on spaced word matches

Robin William Hundt

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

08. June 2020

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Burkhard Morgenstern
Second Supervisor: Dr. Peter Meinicke

I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 08. June 2020

Abstract

Motivation

The amount of sequencing data that is generated is ever increasing. A multiple alignment of these sequences is a NP-complete problem, but constitutes an indispensable part of biology and bioinformatics research. Thus, the need for new heuristics and algorithms able to produce biologically plausible multiple sequence alignments with constrained time and memory resources arises.

In the space of alignment-free sequence comparisons, the concept of Spaced Word Matches proved to be a useful tool for the phylogeny reconstruction of large inputs in a fraction of the time and memory a traditional approach would necessitate. In prior work we established that these Spaced Word Matches can be seen as micro-alignments of segments in the input sequences and that it may be possible to efficiently compute a complete alignment by finding and greedily constructing a consistent set of micro-alignments. The aim of this thesis was to further refine, implement and evaluate this alignment approach, in order to answer the question of whether it is possible to ***efficiently*** *compute* ***biologically plausible*** *multiple sequence alignments by greedily aligning spaced word matches.*

Results

This thesis provides an alignment tool called SpaM-Align, referring to ***S****paced* ***M****atches, which is based on the idea of greedily aligning spaced word matches. A crucial part SpaM-Align is a reimplementation and improvement upon GABIOS-LIB, a library providing the EdgeAddition algorithm for the incremental computation of a transitive closure of an alignment graph.*

The tool is evaluated on the BAliBASE version 3 database and compared to Mafft and Dialign, two established alignment programs. I demonstrate that this approach is able to produce alignments orders of magnitude faster than the Dialign (version 2.2) alignment program, which served as an inspiration for SpaM-Align, and in similar time as the state-of-the-art alignment program Mafft (version 7). However the quality of the produced alignments is, in the best case, slightly worse than those produced by Dialign and significantly worse than what is computed by Mafft.

Contents

1 Introduction

Newer, more cost effective and efficient, sequencing methods lead to an ever increasing amount of sequence data being generated [1]. Due to the problem of aligning N sequences optimally being NP-complete [2], there is a strong need for algorithms and heuristics which are able to process large amounts of data faster than current methods. In the space of phylogeny reconstruction this resulted in a new class of methods referred to as "alignment-free" [3–5]. Among them is the *FSWM*, **F**iltered **S**paced **W**ord **M**atches, approach by Leimeister et al. [3]. It is based on the idea to define a pattern of *Match* and *Don't Care* positions which is slid over the input sequences, resulting in spaced words comprised of the residues at the *Match* positions and a wildcard character "*" at the *Don't Care* positions. Two equal spaced words constitute a spaced word match. It is these matches, which can also be viewed as a micro alignment of segments in the input sequences, that are the basis for the alignment algorithm proposed in the preliminary work to this thesis [6]. The hypothesis being: a complete sequence alignment could be established by finding these micro alignments, assigning a score to each of them and then greedily adding them to a growing alignment. Additionally to proposing an algorithm based on spaced word matches, the preliminary work also evaluated the feasibility of it by examining the distribution of matches on the BAliBASE dataset with respect to their score. As can be seen in figure 1.1, a high scoring spaced word match is likely to be correct while those with a low score are most often incorrect. The "correctness" of these matches representing micro alignments was defined in terms of known *core blocks*, necessitating the third option "Unknown" for an alignment of sites of which neither one is part of a core block.

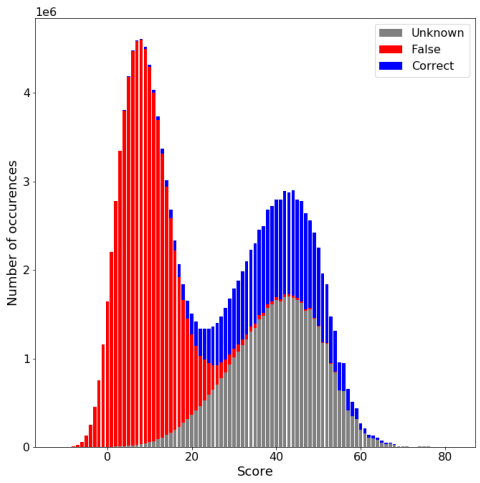

Figure 1.1: Spaced word match histogram for pattern set with weight three and seven *Don't Care* positions on BAliBASE data. Source: [6]

2 | Basics

Understanding the proposed multiple sequence alignment algorithm naturally demands an understanding of the term. While it is usually defined as a generalization of the pairwise sequence alignment problem centered on the insertion of gap characters 2.1, Morgenstern defined an alignment as a consistent equivalence relation 2.2.

2.1 Multiple sequence alignment

As a generalisation of pairwise sequence alignments, multiple sequence alignments are the basis for numerous further analyses such as "inferring phylogenetic relationships, homology search of functional elements, classification of proteins, designing detection markers" [2, pg. 3]. In contrast to the pairwise alignment problem, aligning an arbitrary number of sequences is a NP-complete problem, when formulated as the maximisation (or minimisation) of an objective function [2, pg. 172].
The classical formulation of the problem is based upon a model of evolution where single residues get inserted, deleted or substituted. Since an insertion in one sequence is indistinguishable of a deletion in another one, these two operations are commonly viewed as one and referred to as an *indel*. The goal is then to insert gaps, representing an indel and denoted as '-' into the sequences, such that they have the same length and a given score function is maximal for the produced Alignment. These aligned sequences are usually displayed as a table of residues and gap characters as seen in table 2.1 [2].

seq1	L	L	I	R	N	L	I	Q	V	-	V	K	S	V	-	-	-	-
seq2	L	L	I	R	K	L	I	D	V	-	V	R	T	V	-	-	-	-
seq3	L	L	I	R	Q	L	I	D	V	-	I	K	T	V	-	-	-	-
seq4	L	L	I	-	-	-	-	Q	M	A	D	Q	Y	L	P	E	T	L

Table 2.1: Example of multiple sequence alignment. The gap symbol '-' represents an insertion or deletion (often combined as *indel*).

2.2 Definition of Consistency and Alignments

A more formal definition of the term *Alignment* is based upon the work by Morgenstern et al. [7] and Abdeddaïm [8]. The following definitions constitute a condensed formalization of the one provided in the preliminary work to this thesis [6].

Let S_i be a Sequence over an Alphabet, e.g. DNA, Amino Acids, etc., and $S := \{S_1, ..., S_n\}$ a set of Sequences. The length of the *i-th* sequence is denoted by $len(S_i)$.

Definition 2.2.1 (Site)
A site $x := [i, p]$ represents the p-th position in the i-th sequence and X is the set of all sites for S.
The function

$$seq : X \to \mathbb{N}$$
$$x \mapsto i$$

maps a site to its corresponding sequence. Whereas the function

$$pos : X \to \mathbb{N}$$
$$x \mapsto p$$

maps a site to its position.

Definition 2.2.2 (Ordering of Sites)
For $x = [i, p], x' = [i', p'] \in X$ we define $x \preceq y$ if and only if $i = i'$ and $p \leq p'$.
The relation $\preceq$ is a partial ordering on X.

Lemma 2.2.1 (Extension of binary relation to quasi order relation)
Let A be a reflexive binary relation on some set X and R any binary relation on X.
The transitive closure $\preceq_R := (A \cup R)^t$ is a quasi order relation on X.

As per lemma 2.2.1 we extend $\preceq$ to the quasi partial order $\preceq_R$ by taking the transitive closure of the union of $\preceq$ and R. Consequently $u \preceq v$, vRw, $w \preceq x$, xRy and $y \preceq z$ from which follows that $y \preceq_R z$.

Definition 2.2.3 (Consistency)
Let R be a binary relation on a set of sites X. R is consistent if for $x, y \in X$ where $seq(x) = seq(y)$

$$x \preceq_R y \implies x \preceq y$$

holds. Additionally a set $\{R_1, ..., R_n\}$ of binary relations on X is consistent if $\cup_i R_i$ is consistent.

Definition 2.2.4 (Alignment)
An alignment (or partial alignment) A is a consistent equivalence relation on the set of sites X for a set of sequences S.

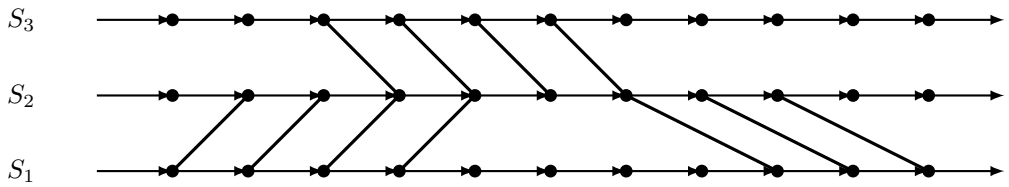

Figure 2.1: Example of an alignment graph representing a consistent equivalence relation A on the set of sites X for $S = \{S_1, S_2, S_3\}$. Undirected connections between nodes x, y represent a relation xAy.

3 | Prior Work

Spam-Align align builds upon several other algorithms and methods, namely GABIOS-LIB by Abdeddaïm [9], Dialign by Morgenstern et al. [7] which later incorporated GABIOS-LIB [8] as well as Spaced Word Matches proposed by Leimeister et al. in *Fast alignment-free sequence comparison using spaced-word frequencies* [3].

3.1 GABIOS-LIB

GABIOS-LIB is a library written by Saïd Abdeddaïm implementing the *EdgeAddition* algorithm for the incremental computation of the transitive closure of a directed cyclic graph for which a spanning set of non overlapping paths is known [9]. In the context of sequence alignment each residue is a node, every node in sequence has a directed edge towards the node corresponding to the residue with the next highest position (thus the sequences form the spanning set of the graph) and an undirected edge between nodes of different sequences equates an alignment of these residues.
Given k sequences with a total length of $n := |X|$ the upper bounds for the computation of a transitivity frontier of a multiple sequence alignment are $O(k^2n + n^2)$ time and $O(kn)$ space.
This is accomplished by incrementally maintaining the transitive closure $\preceq_A$ with A being an equivalence relation on the set of sites $|X|$ defined as $xAy := x$ is aligned to y for $x, y \in X$. Given the quasi partial order $\preceq_A$ an alignment of the sites $x, y \in X$ is consistent iff $x \preceq_A y \iff y \preceq_A x$. Either $x \preceq_A y$ and $y \preceq_A x$ both hold, in which case they are already aligned, or neither of those is true, resulting in $A \cup (x, y)$ being consistent.

Definition 3.1.1 (Transitivity frontiers as defined in [8])
For a given alignment A (as an equivalence relation on a set of sites X), a site $x \in X$ and a sequence index i

we define the predecessor and successor transitivity frontiers in the following way:

$$pred_A[x,i] := \begin{cases} \max\{p : [i,p] \preceq_A x\} & \textit{if there exists a site } [i,p] \textit{ which is } \preceq_A x \\ 0 & \textit{otherwise} \end{cases}$$

$$succ_A[x,i] := \begin{cases} \min\{p : x \preceq_A [i,p]\} & \textit{if there exists a site } [i,p] \textit{ for which } x \preceq_A [i,p] \textit{ holds} \\ len(S_i)+1 & \textit{otherwise} \end{cases}$$

Accordingly $\preceq_A$ can also be defined as $x, y \in X$ with $x = [i,p] : x \preceq_A y \iff p \leq pred_A[y][i]$.

EdgeAddition 1 is able to efficiently maintain this transitive closure by computing the predecessor and successor transitivity frontiers 3.1.1. It is based on the observation that for a site u in a sequence S_i the frontiers towards another sequence S_j, before and after the addition of a consistent site pair (x, y), are related in the following way:

$$pred_{A'}[u][j] = \begin{cases} max\{pred_A[u][j], pred[x][j]\} & \text{if u was successor of y} \\ pred_A[u][j] & \text{otherwise} \end{cases} \tag{3.1}$$

$$succ_{A'}[u][j] = \begin{cases} min\{succ_A[u][j], succ_A[y],[j]\} & \text{if u was predecessor of x} \\ succ_A[u][j] & \text{otherwise} \end{cases} \tag{3.2}$$

Algorithm 1 ensures observation 3.2 by iterating over each pair of sequence indices (i, j) and, in the case of the successor frontier, iterating the sites in sequence S_i from $pred_A[x][i]$ in decreasing order, as long as $succ_A[y][j]$ is smaller than $succ_A[u][j]$, assigning $succ_A[y][j]$ to $succ_A[u][j]$ while it is the minimum of the two. Underlying is the idea, that only those frontier values from sequence i to sequence j are susceptible to change by the alignment of (x, y), which have a path to x (due to this the sites in sequence i are iterated starting at $pred_A[x][i]$ being the highest position in sequence i that has a path to x). Furthermore once a site in sequence i has a $succ_A[u][j]$ value that is **not** greater than $succ_A[y][j]$, there are no further sites in sequence i for which that condition would be true and the next sequence pair can be considered. Updating the predecessor frontier is done in a similar fashion.

As an example we consider the situation displayed in Figure 3.1. The site pair (x, y) is to be aligned and the transitivity frontiers updated. Examining the successor frontier from $i = 2$ towards $j = 1$, the *EdgeAddition* algorithm first considers the predecessor of x in i, namely $pred_A[x][i] = 5$, and since the successor frontier $succ_A[[2, 5][j] = len(S_2)$ is set to the maximum value, it is changed to $succ_A[y][j] = 8$. Since the successor frontier for $[2, 5]$ was updated, the next site in S_2 by decreasing order is considered and the frontier is updated. This continues until site $[2, 2]$ which is aligned to

Algorithm 1: EdgeAddition(x, y) as proposed by Abdeddaïm [9]

Data: Consistent site pair $x, y \in X$ to align
Data: Partial Alignment A as an equivalence relation on X
Data: $succ_A$ successor frontiers for A
Data: $pred_A$ predecessor frontiers for A
Result: Partial Alignment $A' = A \cup \{(x, y)\}$
Result: Updated transitivity frontiers

```
// Update the successor frontier
for i ← 1 to N do
    for j ← 1 to N do
        for p ← pred_A[x][i] to 1 do
            u ← Site[i, p]
            if not succ_A[y][j] < succ_A[u][j] then
                break
            succ_A[u][j] ← succ_A[y][j]
// Update the predecessor frontier
for i ← 1 to N do
    for j ← 1 to N do
        for p ← succ_A[y][i] to len(S_i) do
            u ← Site[p, i]
            if not pred_A[x][j] > pred_A[u][j] then
                break
            pred_A[u][j] ← pred_A[x][j]
```

$[1, 2]$ and thus has a successor frontier of $succ_A[[2, 2]][1] = 2$ which is less the one from y to S_1. Due to the successor frontier from one sequence towards another monotonically falling with decreasing position, the inner for loop can be broken out of and the next sequence pair can be considered.

3.2 Dialign

Dialign is a multiple sequence algorithm first proposed by Morgenstern et al. in 1996 [7] with several alterations and reimplementations over the years [8, 10–13]. In the following, *Dialign* refers to the version 2.2 [13] of the software.

Foremost a pairwise sequence alignment consisting of a set of non overlapping segment pairs (referred to as diagonals) with a maximal sum of scores is computed for every sequence combination. These diagonals are then sorted by their weight (which is computed from the score) and greedily incorporated into an alignment, employing GABIOS-LIB to efficiently ensure its consistency. Diagonals with a weight of 0 are not added to the alignment and discarded. This process of finding diagonals and adding them to an alignment is repeated for the unaligned parts of the sequences until all residues are aligned or no diagonals with a positive score can be found.

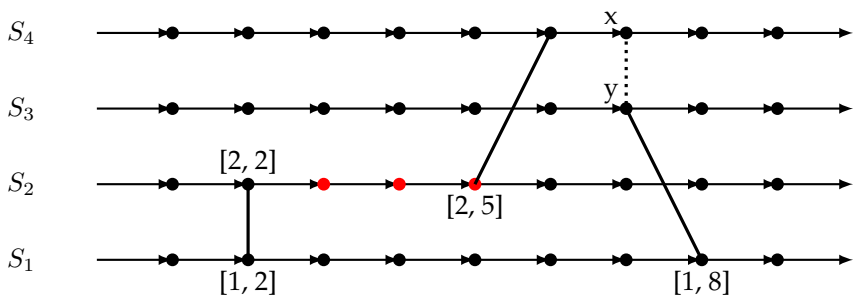

Figure 3.1: Alignment graph visualizing for which nodes (highlighted in red) in sequence $i = 2$ with respect to sequence $j = 1$ the successor frontiers change due to the alignment of (x, y).

Those diagonals which are added first to the alignment establish a *frame* into which the following diagonals need to fit, the expectation being that those diagonals with a very high weight are very likely to be correct. An incorrectly aligned diagonal during those first might lead to the whole alignment being implausible.
Considering this, Dialign can be classified as an iterative and greedy alignment algorithm [7].

3.3 Spaced Word Matches

Spaced word matches are an idea first proposed by Boden et al. [14] (at that time they were referred to as *spaced k-mers*), based upon the idea of *spaced seeds* for fast database searching by Ma et al. [15] and later heavily expanded upon by Chris Leimeister [3–5].
The idea essentially is to search for *inexact* k-mer matches, which can differ at predefined positions, referred to as *"Don't care positions"*.

- based on spaced seeds by [15]
- pattern of care and don't care positions is used to find imprecise? matches between sequences

S_1:	a	b	l	l	h	i	a	f	c	b
S_2:	c	b	l	i	g	i	k	f	i	t
P:		1	1	0	0	0	0	1		

Table 3.1: Example of a Spaced Word Match

Definition 3.3.1 (Pattern as defined in [6])
A Pattern is a sequence over the Alphabet $\Sigma = \{0, 1\}$*, where* 1 *corresponds to a "Match position" and* 0 *to a "Don't Care" position. A patterns weight* k *is defined as the number of "Match positions" it contains.*

4 Algorithm

This thesis provides an implementation and improvement of the alignment algorithm proposed in [6]. It shares the idea of finding segment pairs, or micro alignments, in the input sequences which are greedily aligned with Dialign. However in contrast to that approach, these micro alignments are found by searching for spaced word matches with multiple patterns, a process that is much faster than doing a pairwise alignment for all input sequence combinations.

Algorithm 2: align(S, P)

```
   Data: Sequences S = S_1, ..., S_N
   Data: Pattern set P = P_1, ..., P_M
   Result: Partial Alignment A
 1 A ← {}
 2 micro_alignments ← find_spaced_word_matches(S, P)
 3 micro_alignments.sort_by_score_descending()
 4 foreach ma in micro_alignments do
 5     if is_inconsistent(ma, A) then
 6         continue
 7     foreach site_pair in ma do
 8         if A.not_aligned(site_pair) then
 9             A.add_site_pair(site_pair)
10 insert_gaps(S, A)
11 return S
```

4.1 Finding spaced word matches

TODO: describe find_spaced_word_matches

4.2 Incrementally maintaining a transitive closure

The, unfortunately incomplete, description of the *EdgeAddition* algorithm in [8] lead to the proposal of the algorithm 3 in the preliminary work [6]; due to the incomplete understanding it is

computationally much more expensive than necessary since for every site pair that is added, many sites are considered whose transitivity frontiers can not be influenced.

Algorithm 3: add_site_pair(x, y) as proposed in [6]

Data: Consistent site pair $a, b \in X$ to align
Data: Partial Alignment A
Result: Partial Alignment $A' = A \cup \{(a, b)\}$

```
// Clone the old pred and succ values
```
1 $pred \leftarrow pred_A$
2 $succ \leftarrow succ_A$
```
// Update the successor frontier
```
3 **foreach** $x \in X$ **do**
4 **for** $i \leftarrow 1$ **to** N **do**
5 **if** $x \preceq_A a$ **then**
6 $succ_A[x, i] \leftarrow \texttt{min}(succ[x, i], succ[b, i])$
7 **else if** $x \preceq_{A_i} b$ **then**
8 $succ_A[x, i] \leftarrow \texttt{min}(succ[x, i], succ[a, i])$
9 **else**
10 $succ_A[x, i] \leftarrow succ[x, i]$
```
// Update the predeccessor frontier
```
11 **foreach** $x \in X$ **do**
12 **for** $i \leftarrow 1$ **to** N **do**
13 **if** $x \succeq_A a$ **then**
14 $pred_A[x, i] \leftarrow \texttt{max}(pred[x, i], pred[b, i])$
15 **else if** $x \succeq_{A_i} b$ **then**
16 $pred_A[x, i] \leftarrow \texttt{max}(pred[x, i], pred[a, i])$
17 **else**
18 $pred_A[x, i] \leftarrow pred[x, i]$

Algorithm 4 represents a considerably improved version of 3. It is based on, and improves, the version of *EdgeAddition* contained in Dialign2.2. The most significant improvement over 1 stems from the fact that the transitivity frontiers for sites, which are aligned, coincide, as pointed out in [8]. In other words, given $x \in X$ and an alignment A the following holds $\forall x' \in [x]_A, \forall i \in 1, ..., N$:

$$pred_A[x'][i] = pred_A[x][i] \text{ and}$$
$$succ_A[x'][i] = succ_A[x][i]$$

This allows faster updates and a more compact representation for the transitivity frontiers, since only one value per equivalence class needs to be kept in memory. For those sites x which are not yet aligned, $|[x]_A|$ is 1, the transitivity frontiers are equal to its nearest aligned site.

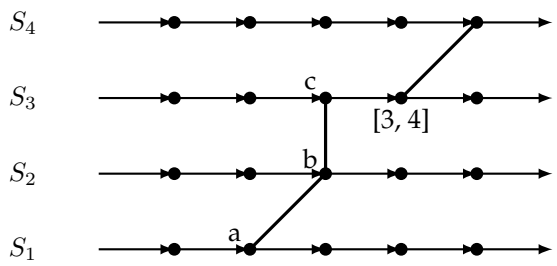

Figure 4.1: Since a, b, c are aligned, their shared successor frontier towards sequence 3 is at position 4.

4.3 Inserting gaps to produce an alignment from transitivity closure

Once the list of spaced word matches that are found in the sequences S is filtered and added to the transitive closure, the information about the equivalence classes of aligned sites can be utilized to efficiently insert gaps into the input sequences such that those sites belonging to the same class are in the same column of the resulting alignment.

This can be achieved by using the quasi-order $\preceq_A$, originally defined on the set of sites X, as a partial order on the alignment equivalence classes $[x]_A$. This is allowed by the simple observation that $\forall x, y \in X, x' \in [x]_A$ and $y' \in [y]_A$, $x' \preceq_A y'$ if and only if $x \preceq_A y$ which leads to the definition $[x]_A \preceq_A [y]_A := x \preceq_A y$. Anti-symmetry follows trivially and thus $\preceq_A$ partially orders the alignment equivalence classes.

Since the revised *EdgeAddition* algorithm 4 already stores information about equivalence classes, it is easy to collect a partially ordered list of equivalence classes E. Using this sorted list, the sequences can be aligned by taking the first class, setting the positions of the aligned sites to the maximum position in the class and then shifting the sites of the eq classes coming after the aligned one by the number of gaps that were added to the sequences.

Algorithm 4: revised add_site_pair(x, y) based on GABIOS-LIB implementation in Dialign2.2 [8]

Data: Consistent site pair $x, y \in X$ to align
Data: Index nn of alignment set that was merged from alignment sets of x and y
Data: Successor and predecessor frontiers $succ, pred$
Data: $pred_x$ and $pred_y$ are predecessor frontiers of x and y respectevily
Data: $alig\_set$ matrix, mapping an alignment set and sequence to a position
Data: $pred\_alig\_set\_pos$ mapping a sequence and position to the index of the next predecessor alignment set of that site, $succ\_alig\_set\_pos$ equivalent for successor alignment set
Result: Updated tranisitivity frontiers $succ$ and $pred$

```
// Calculate updates to successor frontier
succ_frontier_ops ← [ ]
for i ← 1 to N do
    if pred_x[i] == pred_y[i] then
        continue
    for j ← 1 to N do
        k ← pred[nn, i]
        if k > 0 and k == alig_set[nn, i] then
            k ← pred_alig_set_pos[i, k]
        while k > 0 do
            n ← alig_set_nbr[i, k]
            if succ[n, j] > succ[nn, j] then
                succ_frontier_ops.push ([n, j, succ[nn, j]])
                k ← pred_alig_set_pos[i, k]
            else
                break
// Calculate updates to predecessor frontier
pred_frontier_ops ← [ ]
for i ← 1 to N do
    if succ_x[i] == succ_y[i] then
        continue
    for j ← 1 to N do
        k ← succ[nn, i]
        if k == alig_set[nn, i] then
            k ← succ_alig_set_pos[i, k]
        while k > 0 do
            n ← alig_set_nbr[i, k]
            if pred[n, j] < pred[nn, j] then
                succ_frontier_ops.push ([n, j, pred[nn, j]])
                k ← succ_alig_set_pos[i, k]
            else
                break
foreach [n, j, new_front] ∈ succ_frontier_ops do
    succ[n, j] ← new_front
foreach [n, j, new_front] ∈ pred_frontier_ops do
    pred[n, j] ← new_front
```

5 | Implementation

The implementation of the algorithm is done in the Rust programming language [1] and contained in the git sub module `spam-align` of the alignment evaluation folder.
The core part of the algorithm as described in 4 is designed to iteratively maintain a partial alignment, as per definition 2.2.4, that allows the fast insertion of newly aligned sites as well as to check if a given pair of sites is consistent with the current alignment.
While an initial implementation, called GABIOS-LIB, is part of the *Dialign2.2* program, *spam-align* contains an improved reimplementation of that library.
Although the algorithm as described in 4 has the same asymptotic time and memory complexity as the initial implementation, considerable effort was spent on improving the actual implementation by simplifying the code, increasing the cache friendliness, reducing unnecessary allocations and generally enhancing the performance.

```
pub struct Closure {
 sequences: Sequences,
 alig_set: Matrix<usize>,

 nbr_alig_sets: usize,
 old_nbr_alig_sets: usize,

 pred_frontier: Matrix<usize>,
 succ_frontier: Matrix<usize>,

 pred_frontier_ops: Vec<FrontierOp>,
 succ_frontier_ops: Vec<FrontierOp>,
}

struct Sequences {
  lengths: Vec<usize>,
  alig_set_nbr: Matrix<usize>,
  pred_alig_set_pos: Matrix<usize>,
  succ_alig_set_pos: Matrix<usize>,
}
```

Listing 1: Data types responsible for storing transitive closure.

Listing 1 provides the definition of the core data types responsible for tracking the status of an alignment that is constructed by iteratively aligning sites. A `struct` in Rust functions as a simple record of heterogeneous data similar to those in the *C* programming language. Members of a `struct` are defined as `<name of member field>: <type of field>`.

[1] rust-lang.org/

In contrast to GABIOS-LIB, a dedicated `Matrix` type is used which is backed by a contiguous section of memory and not a pointer to memory containing pointers to the actual data. Allocating a matrix of n rows thus only takes a single allocation instead of $n+1$, additionally reducing pointer indirection and increasing CPU cache utilization.
For the algorithm 4 the predecessor frontiers of x and y are needed, so that sequences j, for which there is no possibility of change, can be skipped. Distinguishing the improved algorithm is the detail that these frontiers do not need to be computed for each insertion of a site pair and stored in a separate data structure; rather it is sufficient, and faster, to simply compute the appropriate alignment set index and provide $pred_x$ as a pointer to the right row of $pred$.
A key difference and major improvement is constituted by the way updates to the frontiers are applied. Since line 11 introduces a data anti-dependency,meaning a value must be written after it is read, between successive iterations, changing the transitivity frontiers can not be done in place. In the improved version `FrontierOperations` consisting of the alignment set index, the sequence and the new frontier value are stored in a growable memory buffer, called a `Vec`, and applied to the frontiers afterwards. Although applying the changes in this way might require more memory than storing the new values in a $N \times N$ matrix and computing the requisite indices for the $pred$ structure, it does not lead to asymptotically more memory consumption and has the benefit of being considerably faster.
GABIOS-LIB implements the member `sequences` of the `Closure` struct as a vector of structs containing vectors, further introducing overhead by pointer indirection. As can be seen in listing 1, this structure is changed to a struct of matrices, reducing the number of pointers that need to be followed in order to access a value in e.g. `pred_alig_set_pos` from two to one.
During the development process, benchmarks located at `spam-align/benches` were used to ensure that the described optimizations do in fact improve the performance of the algorithm.

6 Evaluation

To answer the research question of whether a multiple sequence alignment algorithm based solely on spaced word matches and consistency checking is able to **efficiently** provide **biologically plausible** alignments, we need to systematically evaluate and compare it to different approaches. Asserting whether the algorithm is efficient can be done by simply running it on different input sizes and comparing it to other alignment programs. However, determining whether the produced alignments are biologically plausible in order to be used in further analysis is a much harder task and in itself an active area of research [16]. This stems from the fact that an evaluation of alignment algorithms, just like its implementation, must make simplifying assumptions about the underlying evolutionary processes. There simply is no way of *"proving"* an alignment algorithm is *"correct"* or even *"better"* than others, since we can not know the *"true"* alignment of a sequence family. However, this does not mean that we cannot make reasonable assumptions to base an evaluation on.

6.1 BAliBASE 3 alignment benchmark dataset

The third version of the BAliBASE benchmark protein alignment database has been released in 2005 and is widely employed for the comparison of multiple alignment programs [2, 17]. It is constructed in a semi-automatic process, as shown in fig. 6.1 and suitable to evaluate global and local alignment programs. The database is split into five reference sets with different characteristics representing distinctive multiple alignment problems.
It is divided into:

- reference set 1 subset V1, for which any two sequences share <20% identity and no internal insertions over 35 residues long

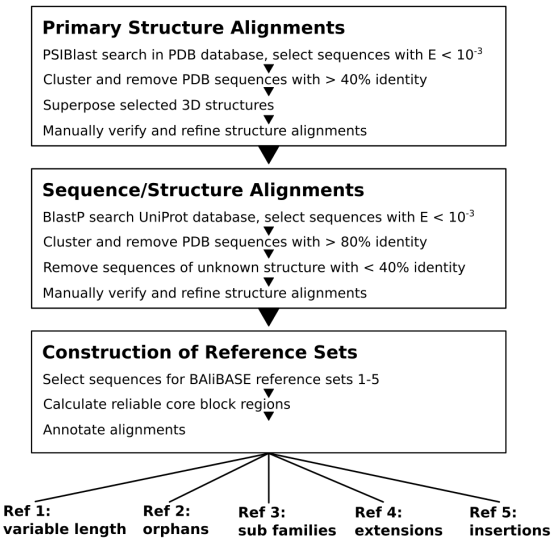

Figure 6.1: Semi automatic process used to establish the reference sets. Source: [6]

- reference set 1 subset V2, consisting of families with at least four equidistant sequences for which any two sequences share 20-40% identity and no large insertions
- reference set 2, for which all sequences share >40% identity and at least one 3D structure is known. Additionally, an "Orphan" sequence with <20% identity is chosen per family
- for reference set 3, all sequences in the same subfamily have >40% identity, whereas sequences from different subfamilies share <20% identity
- for reference sets 4 and 5, every sequence shares at least 20% with one other sequence, including sequences with large N/C-terminal extensions (ref 4) or internal insertions (ref 5)

6.1.1 Core blocks

Evaluating and comparing alignment programs is a challenging problem due to the uncertainty of supposedly "real" alignments of actual sequences. The BAliBASE database marks alignment columns which can be reliably aligned as so-called "core blocks". These core blocks are calculated and manually verified, making up 19% of the full-length sequences which are used in the evaluation of *spam-align* [17].

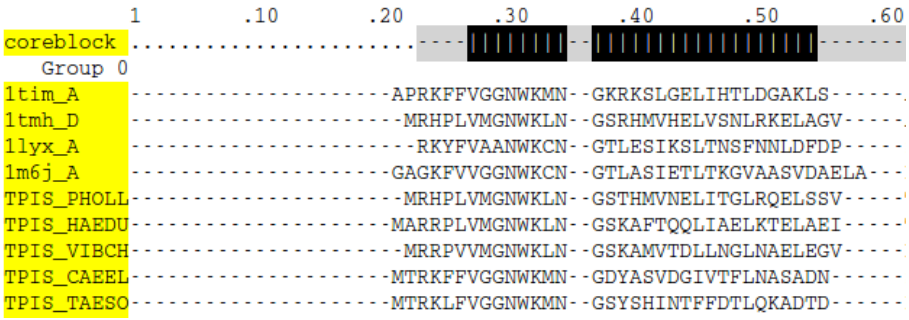

Figure 6.2: BAliBASE web interface. Black columns indicate core blocks. lbgi.fr

6.2 Sum-of-pairs and column score

Comparing the alignment output of different methods can be done by computing the sum-of-pairs and column scores.

Given a test alignment A_t and a reference alignment A_r with M sequences and N_t, N_r columns respectively, the sum-of-pairs and column score is defined according to Thompson et al. [18].

Definition 6.2.1 (Sum-of-pairs score)
The sum of pairs score is the ratio of correctly aligned individual residues. Formally it is defined as:

$$p_{ijk} = \begin{cases} 1 \textit{ if residues } A_{t_{ij}} \textit{ and } A_{t_{ik}} \textbf{\textit{ are }} \textit{aligned in } A_r \\ 0 \textit{ otherwise} \end{cases}$$

$$S_i = \sum_{j=1}^{M} \sum_{k=i+1}^{M} p_{ijk}$$

$$SPS = \frac{\sum_{i=1}^{N_t} S_i}{\sum_{i=1}^{N_r} S_{r_i}}$$

with S_{r_i} being the number of correctly aligned residues in the reference.

Definition 6.2.2 (Column score)
The column score is the ratio of correctly aligned columns.

$$C_i = \begin{cases} 1 \textit{ if all the residues in the i-th column are aligned correctly} \\ 0 \textit{ otherwise} \end{cases}$$

$$CS = \frac{\sum_{i=1}^{N_t} C_i}{N_r}$$

Note that $C_i = 1$ only if all the residues in the i-th column are aligned correctly and no residue belonging to this column is part of another one. For this reason, the numerator is smaller or equal to the denominator.
The definition of the column score is slightly different from that provided by the authors of BAliBASE [18] but resembles the actual implementation in the included BaliScore tool and its reimplementation provided with this thesis.
These scores are only calculated for the core blocks of the BAliBASE alignments, meaning that for the following evaluation A_t is an alignment over the full sequences, while A_r contains only the aligned residues inside the core blocks.

6.3 Bali-Score

Accompanying this thesis is a reimplementation of the *bali-score* tool, which can be used to calculate the SPS and column scores given a test alignment in `fasta` format and a BAliBASE reference `xml` file. Although some of the features which are part of the original implementation are not contained in this version, it should be significantly easier to use and install since the correct dependencies are

automatically resolved and fetched by the Rust package manager.
The source code is available on github.com and, provided the current stable Rust toolchain [1] is available, installing the tool can simply be done by issuing `cargo install -path .` in the cloned repository root. For usage instructions view `bali-score -h`.

6.4 Evaluated programs

Answering the research question of whether the proposed approach is able to compute alignments *fast* while being *biologically plausible*, it is necessary to compare it against other established approaches. *SpaM-Align* (where SpaM stands for **S**paced **M**atches) is the alignment program produced as part of this thesis. Although *Dialign* is a dated approach, version 2.2 was released in 1999, it heavily influenced *SpaM-Align* and therefore provides an interesting comparison. The third program we compare, *Mafft*, is a state of the art multiple sequence alignment tool which enables the user to choose and mix progressive, iterative and structural alignment strategies.

6.4.1 SpaM-Align

SpaM-Align is the alignment tool which implements the greedy algorithm described in chapter 4. The source code is openly available[2] and the tool can be installed by executing `cargo install -path .` in the project root (see `README.md` for further details). A description of the available arguments can be displayed with `spam-align -help`.

Evaluated patterns

Being based on spaced word matches, *SpaM-Align* requires one or more patterns 3.3.1 as a parameter. In earlier work we established that it is potentially possible to identify a high ratio of homologous sites correctly, however choosing an unfit pattern set with a weight that is too high, fails to identify a large number of homologous sites. To better understand the influence that a patterns weight and the number of *Don't Care* positions has, we evaluate *SpaM-Align* on 84 different pattern sets. *Rasbhari* by Hahn et al. [19], a tool to generate and optimize patterns for alignment-free sequence comparison, is used to create a pattern set for every combination of weight $k \in 2, ..., 7$ and number of *Don't Care* positions $d \in 3, ..., 16$ with the following additional parameters applied:

```
-m 5        # patterns per set
-q 0.05     # background match probability
-p 0.3      # match probability
-S 400      # sequence length
-H 20       # homologue region length
```

[1] View rustup.rs for installation instruction
[2] Source available at: github.com/robinhundt/spam-align

Since *Rasbhari* always places a *Match* position at the beginning and end of a pattern, for a weight of 2 only patterns of the form 10^d1 are evaluated.

6.4.2 Dialign

Dialign, described under prior work 3.2, was executed without additional parameters.

6.4.3 Mafft

Mafft (version 7), standing for **m**ultiple **a**lignment using **f**ast **F**ourier **t**ransform, is a widely used similarity based multiple sequence alignment program employing progressive, iterative and structural strategies [20].
Among the many different alignment options offered by *Mafft*, we chose FFT-NS-1 and L-INS-i as evaluation targets. These two represent the extremes of the accuracy and speed trade-off, with FFT-NS-1 being labelled as "very fast; recommended for >2000 sequences" and L-INS-i as "probably most accurate; recommended for <200 sequences". FFT-NS-1, from now on referred to as *Mafft-Fast*, is a progressive method utilizing an imprecise guide tree; L-INS-i, referred to as *Mafft-Accurate* in the following, constitutes an iterative refinement method which integrates local pairwise alignment information [20].

6.5 Results

Figure 6.3 gives an overview over how different pattern configurations influence the Sum-of-Pairs score of *SpaM-Align* aggregated over the whole BAliBASE dataset and relates those to the mean Sum-of-Pairs score of Mafft-Accurate (0.867), Mafft-Fast (0.801) and Dialign (0.769). It is noteworthy that a higher weight seems to result in a better score with pattern sets of weight 7 performing the worst and sets of weight two being substantially better than those with a weight of four. The best Sum-of-Pairs score for *SpaM-Align*, 0.687, is achieved with a single pattern of weight two and eleven *Don't Care* positions. Interestingly increasing the number of *Don't Care* positions seems to only increase the scores of the produced alignments until $d = 11$ with most of the curves plateauing after this. Unexpectedly the pattern of weight two performs worse than those with three or four *Match* positions for a low number of d.

Similar to figure 6.3, the plot 6.4 displays the column score 6.2.2 for a large number of weight and amount of *Don't Care* positions, aggregated over the complete BAliBASE data and set into context by the values for Mafft-Accurate, Mafft-Fast and Dialign. The column scores for these three points of comparison are 0.617, 0.475 and 0.450 respectively. *SpaM-Align* achieves the highest column score, 0.335, with a pattern of weight two and twelve *Don't Care* positions.

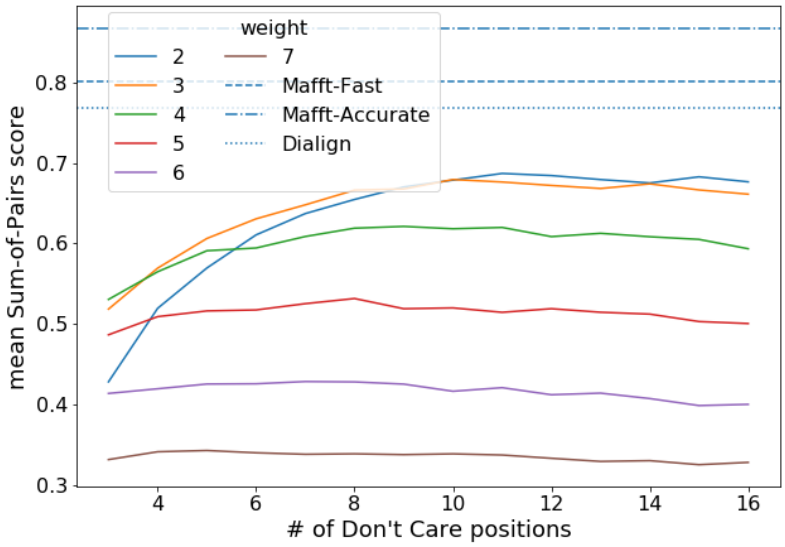

Figure 6.3: Sum-of-Pairs scores aggregated over BAliBASE reference sets and weights and number of *Don't Care* positions.

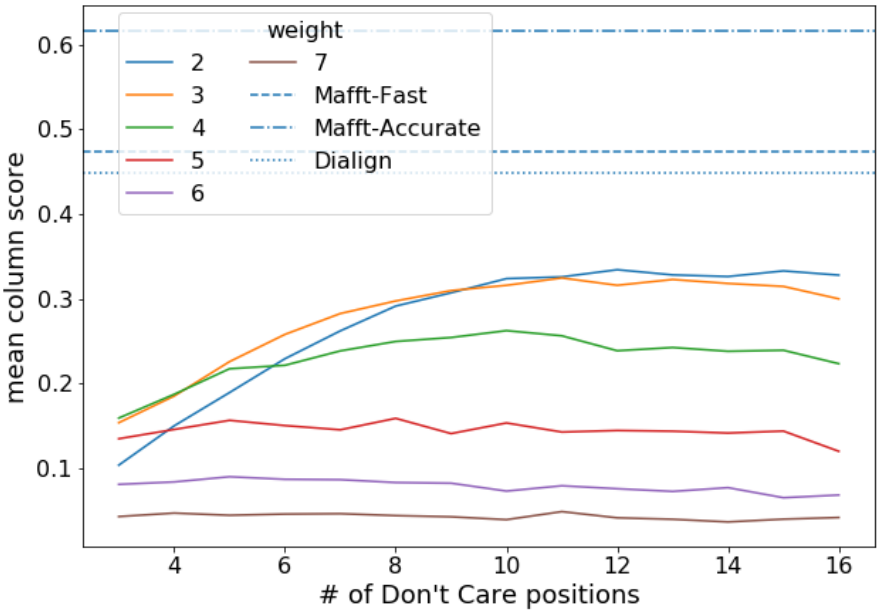

Figure 6.4: Column scores aggregated over BAliBASE reference sets and weights.

	RV11	RV12	RV20	RV30	RV40	RV50
Dialign	0.494482	0.851870	0.868279	0.739987	0.830939	0.804629
Mafft-Accurate	0.648562	0.937168	0.927191	0.862048	0.917403	0.899301
Mafft-Fast	0.521931	0.890052	0.885096	0.812195	0.842169	0.850608
SpaM-Align $k = 2, d = 11$	0.261811	0.751974	0.850375	0.715074	0.789201	0.733125
SpaM-Align $k = 2, d = 13$	0.247997	0.751732	0.838872	0.724464	0.770202	0.733305
SpaM-Align $k = 3, d = 10$	0.250701	0.733341	0.844282	0.714178	0.789304	0.724828

Table 6.1: Mean Sum-of-Pairs scores for compared programs.

	RV11	RV12	RV20	RV30	RV40	RV50
Dialign	0.267504	0.695353	0.320019	0.411190	0.484710	0.496890
Mafft-Accurate	0.438860	0.851518	0.486511	0.657281	0.627038	0.621163
Mafft-Fast	0.264334	0.763283	0.340603	0.467281	0.478159	0.528384
SpaM-Align $k = 2, d = 11$	0.087883	0.548499	0.192057	0.343963	0.404308	0.350156
SpaM-Align $k = 2, d = 13$	0.068096	0.562437	0.222104	0.367169	0.397829	0.289876
SpaM-Align $k = 3, d = 10$	0.075997	0.525526	0.189796	0.334228	0.406251	0.324160

Table 6.2: Mean column scores for compared programs.

But these scores represent the mean over every computed alignment in the BAliBASE database and hide the substantial differences between the different reference sets 6.1. Table 6.1 shows the mean sum-of-pairs scores per subset of BAliBASE for the benchmarked programs (only selected parameters for *SpaM-Align*); table 6.2 shows the mean column scores for the same programs and parameters.

However, even within a reference set of BAliBASE , the Sum-of-Pairs and to a greater extent the column scores are spread widely, as can be seen by the large inter quartile ranges of the box plots 6.5 and 6.6.

6.5.1 Execution times

Looking at figure 6.7, we see that the execution time of *SpaM-Align* is significantly better than that of *Dialign* (mean execution time of 26.5 seconds) and *Mafft-Accurate* (mean of 16.3 second) lying between 100 ms and 500 ms depending on the weight and number of *Don't Care* positions in the provided pattern set. For pattern sets with a weight of three or more, *SpaM-Align* executes even faster than the speed optimized strategy *Mafft-Fast* (mean of 431 ms).
However table 6.3 reveals that although *SpaM-Align* only takes 5ms - 20ms on the reference sets RV11 and RV12 containing smaller sequences, compared 217ms and 254ms for *Mafft-Fast*, the execution time on the larger sequences of RV30 is more than twice as *Mafft-Fast* which aligns them, on average, in 455ms.

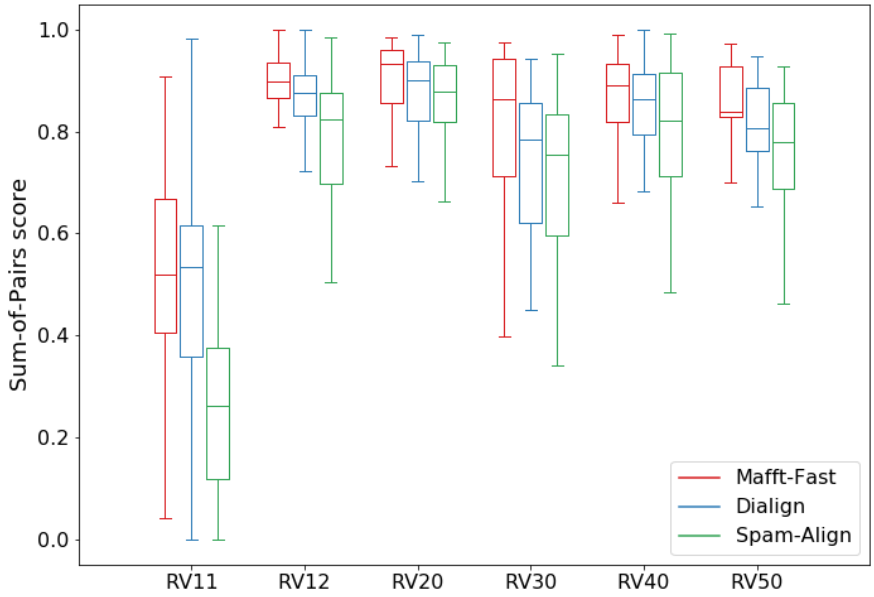

Figure 6.5: Sum of pairs scores. Spam-Align is called with a $k = 2, d = 11$ pattern

	RV11	RV12	RV20	RV30	RV40	RV50
Dialign	543	1626	40116	85923	22494	23141
Mafft-Accurate	892	1276	14676	25992	33743	26462
Mafft-Fast	217	254	425	455	705	559
SpaM-Align $k = 2, d = 11$	5	18	592	1309	270	323
SpaM-Align $k = 2, d = 13$	5	18	609	1338	279	340
SpaM-Align $k = 3, d = 10$	5	15	485	1024	188	222

Table 6.3: Mean execution times for compared programs

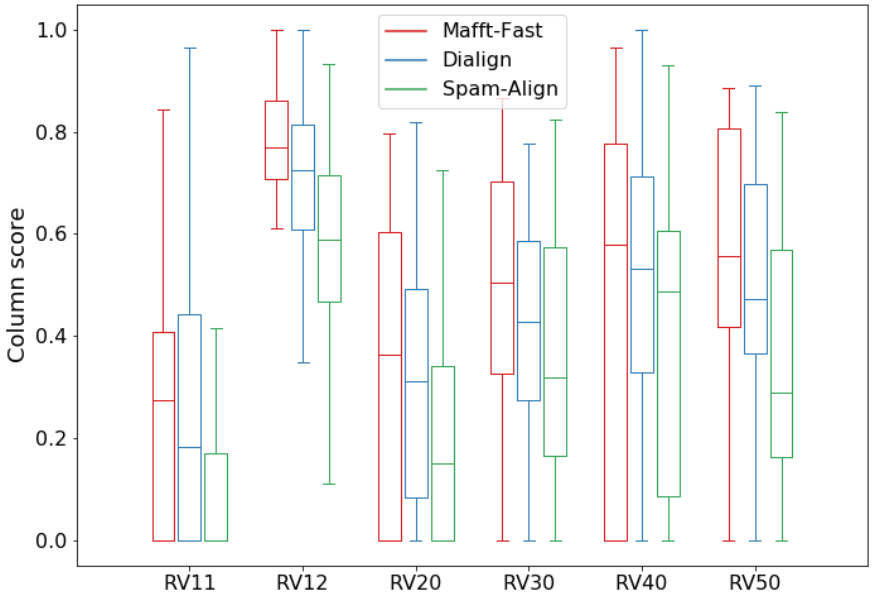

Figure 6.6: Column scores. Spam-Align is called with a $k = 2, d = 11$ pattern.

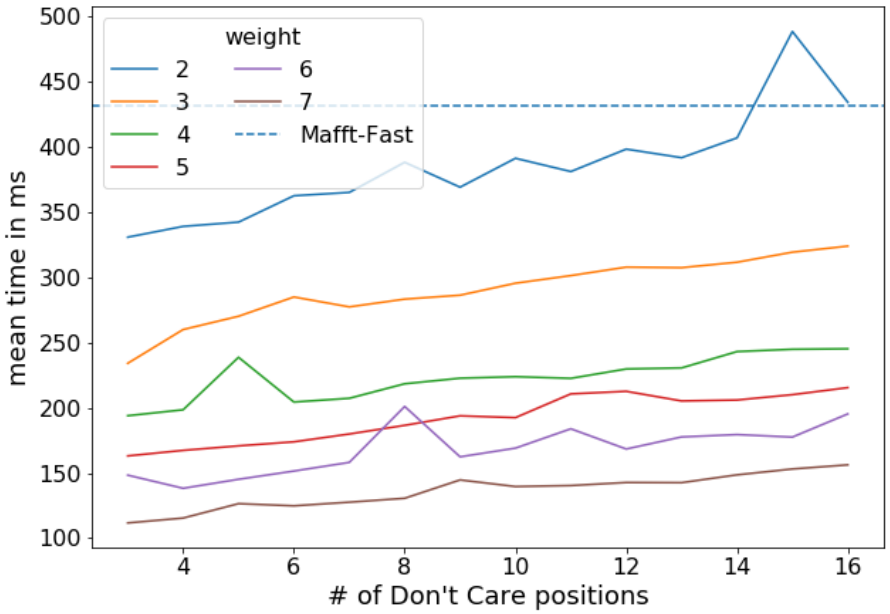

Figure 6.7: Mean execution times in ms.

7 | Discussion

The evaluation and comparison of *SpaM-Align* against *Dialign* and *Mafft* on the BAliBASE dataset revealed that the produced alignments are slightly inferior to Dialign in the best case, using a pattern with two *Match* and eleven *Don't Care* positons on the RV20 reference set, and significantly worse than *Mafft* in every case.

8 Conclusion

8.1 Further work

- combining patterns with different weight ratios
- improve generation of micro alignments compared to current primitive one

8.1.1 Parallelisation Opportunities

Bibliography

[1] P. Muir, S. Li, S. Lou, D. Wang, D. J. Spakowicz, L. Salichos, J. Zhang, G. M. Weinstock, F. Isaacs, J. Rozowsky *et al.*, "The real cost of sequencing: scaling computation to keep pace with data generation," *Genome biology*, vol. 17, no. 1, p. 53, 2016.

[2] D. J. Russell, *Multiple Sequence Alignment Methods* -, softcover reprint of the original 1st ed. 2014 ed. Humana Press, 2016.

[3] C.-A. Leimeister, M. Boden, S. Horwege, S. Lindner, and B. Morgenstern, "Fast alignment-free sequence comparison using spaced-word frequencies," *Bioinformatics*, vol. 30, no. 14, pp. 1991–1999, 2014.

[4] C.-A. Leimeister, S. Sohrabi-Jahromi, and B. Morgenstern, "Fast and accurate phylogeny reconstruction using filtered spaced-word matches," *Bioinformatics*, vol. 33, no. 7, pp. 971–979, 2017.

[5] C.-A. Leimeister, T. Dencker, and B. Morgenstern, "Accurate multiple alignment of distantly related genome sequences using filtered spaced word matches as anchor points," *Bioinformatics*, vol. 35, no. 2, pp. 211–218, 2018.

[6] R. W. Hundt, "Praktikumsbericht," *Vertiefte anwendungsorientierte Systementwicklung im forschungsbezogenen Praktikum, Georg-August-Universität Göttingen*, 2020.

[7] B. Morgenstern, A. Dress, and T. Werner, "Multiple dna and protein sequence alignment based on segment-to-segment comparison," *Proceedings of the National Academy of Sciences*, vol. 93, no. 22, pp. 12 098–12 103, 1996.

[8] S. Abdeddaïm and B. Morgenstern, "Speeding up the dialign multiple alignment program by using the 'greedy alignment of biological sequences library'(gabios-lib)," in *International Conference on Biology, Informatics, and Mathematics*. Springer, 2000, pp. 1–11.

[9] S. Abdeddaïm, "On incremental computation of transitive closure and greedy alignment," in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 1997, pp. 167–179.

[10] B. Morgenstern, "Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment." *Bioinformatics (Oxford, England)*, vol. 15, no. 3, pp. 211–218, 1999.

[11] A. R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern, "Dialign-t: An improved algorithm for segment-based multiple sequence alignment," *BMC Bioinformatics*, vol. 6, p. 66, 2005.

[12] A. R. Subramanian, M. Kaufmann, and B. Morgenstern, "Dialign-tx: greedy and progressive approaches for the segment-based multiple sequence alignment," *Algorithms for Molecular Biology*, vol. 3, p. 6, 2008.

[13] B. Morgenstern, "Dialign: multiple dna and protein sequence alignment at bibiserv," *Nucleic acids research*, vol. 32, no. suppl_2, pp. W33–W36, 2004.

[14] M. Boden, M. Schöneich, S. Horwege, S. Lindner, C. Leimeister, and B. Morgenstern, "Alignment-free sequence comparison with spaced k-mers," in *German Conference on Bioinformatics 2013*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[15] B. Ma, J. Tromp, and M. Li, "Patternhunter: faster and more sensitive homology search," *Bioinformatics*, vol. 18, no. 3, pp. 440–445, 2002.

[16] R. C. Edgar, "Quality measures for protein alignment benchmarks," *Nucleic acids research*, vol. 38, no. 7, pp. 2145–2153, 2010.

[17] J. D. Thompson, P. Koehl, R. Ripp, and O. Poch, "Balibase 3.0: latest developments of the multiple sequence alignment benchmark," *Proteins: Structure, Function, and Bioinformatics*, vol. 61, no. 1, pp. 127–136, 2005.

[18] J. D. Thompson, F. Plewniak, and O. Poch, "A comprehensive comparison of multiple sequence alignment programs," *Nucleic acids research*, vol. 27, no. 13, pp. 2682–2690, 1999.

[19] L. Hahn, C.-A. Leimeister, R. Ounit, S. Lonardi, and B. Morgenstern, "Rasbhari: Optimizing spaced seeds for database searching, read mapping and alignment-free sequence comparison," *PLoS computational biology*, vol. 12, no. 10, p. e1005107, 2016.

[20] K. Katoh and D. M. Standley, "Mafft multiple sequence alignment software version 7: improvements in performance and usability," *Molecular biology and evolution*, vol. 30, no. 4, pp. 772–780, 2013.