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

- *sequencing data is ever increasing*
- *MSA, while being an indispensable part of biology and bioinformatics research is a computationally very expensive problem*
- *in need of new heuristics and algorithms which can be used to calculate MSA with constrained resources*
- *earlier work examined the feasibility of an alignment approach based on spaced word matches and consistency checks in conjuction with GABIOS-LIB*
- *this work provides an implementation and evaluation of this approach*

Results

Contents

1 Introduction

- maybe show spamograms from [1] here which show that high scoring spaced word matches are very likely to be correct as motivation

2 | Basics

- what is a MSA and for what is it important?
- math def of consistency and msa

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. [3] and Abdeddaïm [4]. The following definitions constitute a condensed formalization of the one provided in the preliminary work to this thesis [1].

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.

TODO: Clarify what an alignment graph and a path in one is. Terminology is later used.

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.

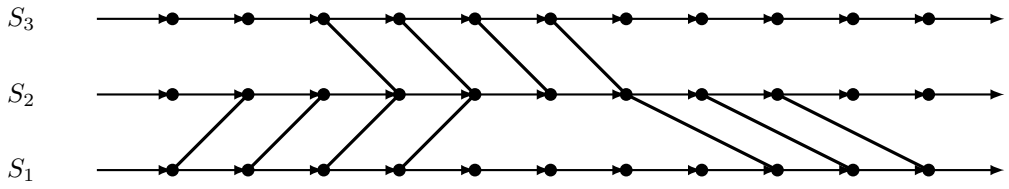

Figure 2.1: Example of an alignment graph representing a consistent equivalence relation A on the set of sites X for $S = \{S_1, S_2, S_3\}$. Undirected connections between nodes x, y represent a relation xAy.

3 | Prior Work

Spam-Align align builds upon several other algorithms and methods, namely GABIOS-LIB by Abdeddaïm [5], Dialign by Morgenstern et al. [3] which later incorporated GABIOS-LIB [4] as well as Spaced Word Matches proposed by Leimeister et al. in *Fast alignment-free sequence comparison using spaced-word frequencies* [6].

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 [5]. 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 [4])
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 [5]

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
```

1 **for** $i \leftarrow 1$ **to** N **do**
2 **for** $j \leftarrow 1$ **to** N **do**
3 **for** $p \leftarrow pred_A[x][i]$ **to** 1 **do**
4 $u \leftarrow Site[i, p]$
5 **if** *not* $succ_A[y][j] < succ_A[u][j]$ **then**
6 **break**
7 $succ_A[u][j] \leftarrow succ_A[y][j]$

```
// Update the predecessor frontier
```

8 **for** $i \leftarrow 1$ **to** N **do**
9 **for** $j \leftarrow 1$ **to** N **do**
10 **for** $p \leftarrow succ_A[y][i]$ **to** $len(S_i)$ **do**
11 $u \leftarrow Site[p, i]$
12 **if** *not* $pred_A[x][j] > pred_A[u][j]$ **then**
13 **break**
14 $pred_A[u][j] \leftarrow pred_A[x][j]$

$[1, 2]$ 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.

- written to be used in greedy alignment algorithms, used in Dialign 2.2
- able to answer the question of whether two sites are alignable in $o(1)$
- allows efficient incremental updates of closure when new sites are added to alignment
- works by incrementally computing transitivity frontiers
- $O(k^2n + n^2)$ time for k sequences with a total length of n ($n = |X|$)
- $O(kn)$ space
- works by maintaining predecessor and successor transitivity frontiers -> define them
- $\preceq_A$ can be defined in terms of pred frontier

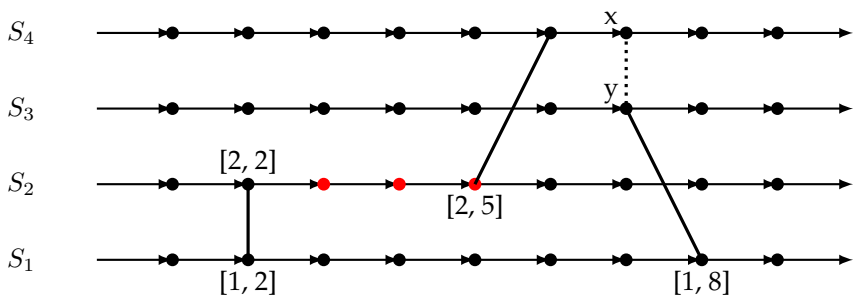

Figure 3.1: Alignment graph visualizing for which nodes (highlighted in red) in sequence $i = 2$ with respect to sequence $j = 1$ the successor frontiers change due to the alignment of (x, y).

3.2 Dialign

Dialign is a multiple sequence algorithm first proposed by Morgenstern et al. in 1996 [3] with several alterations and reimplementations over the years [4,7–10]. In the following, *Dialign* refers to the version 2.2 [10] 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. 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 [3].

- is an iterative and greedy algorithm
- first computes pairwise alignments consisting of a set of non overlapping segment pairs (referred to as diagonals) with a maximal sum of scores for every combination of sequences
- set of diagonals is scored by their weight and greedily incorporated into alignment while using GABIOS-LIB to efficiently ensure consistent of alignments
- diagonals with score below threshold (default 0) are discarded
- process of finding diagonals and adding them to alignment is repeated for unaligned parts of sequences until everything is aligned or no diagonals of positive score can be found
- dialign sepnds majority of time on constructing paiwise alignments in order to find diagonals

-> src is own lousy profile, maybe show profile data for big alignment in bb in appendix?

3.3 Spaced Word Matches

Spaced word matches are an idea first proposed by Boden et al. [11] (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. [12] and later heavily expanded upon by Chris Leimeister [6, 13, 14].
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 [12]
- 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 [1])
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.*

3.3.1 Multi dimensional matches

TODO: Leave this is out in case analysis of multi dim matches is not included in evaluation.

S_1:	a	b	l	l	h	i	a	f	c	b
S_2:	c	b	l	i	g	i	k	f	i	t
S_3:	k	b	l	q	k	i	a	f	i	l
P:		1	1	0	0	0	0	1		

Table 3.2: Example of a multi dimensional Spaced Word Match

4 Algorithm

This thesis provides an implementation and improvement of the alignment algorithm proposed in [1]. 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.

- describe how micro alignments, especially multi dim are found?
- what scoring function should be used and why?
- TODO maybe have a look at overlap weight again
- describe version of algorithm utilizing eq classes and property described in **??** -> should this include data structures and memory evincemanagement?
- maybe first dp algorithm without unnecessary cmps (like in Fi. 4 of [5]) and the formulate with eq classes (which wasn't done in paper)
- highlight how this differs from most alignment algorithms in that it is completely greedy

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 return A
```

TODO: describe find_spaced_word_matches

The, unfortunately incomplete, description of the *EdgeAddition* algorithm in [4] lead to the proposal of the algorithm 3 in the preliminary work [1]; 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 [1]

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
$pred \leftarrow pred_A$
$succ \leftarrow succ_A$
// Update the successor frontier
foreach $x \in X$ **do**
  for $i \leftarrow 1$ **to** N **do**
    if $x \preceq_A a$ **then**
      $succ_A[x, i] \leftarrow \texttt{min}(succ[x, i], succ[b, i])$
    else if $x \preceq_{A_i} b$ **then**
      $succ_A[x, i] \leftarrow \texttt{min}(succ[x, i], succ[a, i])$
    else
      $succ_A[x, i] \leftarrow succ[x, i]$
// Update the predeccessor frontier
foreach $x \in X$ **do**
  for $i \leftarrow 1$ **to** N **do**
    if $x \succeq_A a$ **then**
      $pred_A[x, i] \leftarrow \texttt{max}(pred[x, i], pred[b, i])$
    else if $x \succeq_{A_i} b$ **then**
      $pred_A[x, i] \leftarrow \texttt{max}(pred[x, i], pred[a, i])$
    else
      $pred_A[x, i] \leftarrow pred[x, i]$

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 [4]. 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.

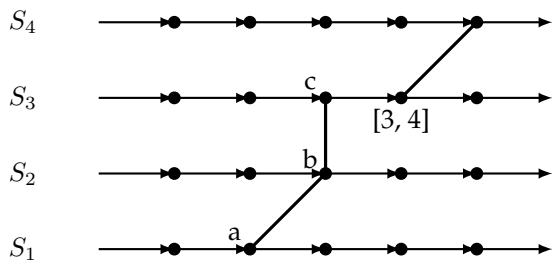

Figure 4.1: Since a, b, c are aligned, their shared successor frontier towards sequence 3 is at position 4.

While the algorithm 3 for maintaining a transitive closure when adding a pair of sites (a, b) into a partial alignment works, it is far from optimal.

TODO: This belongs in prior work since it's part of the original EdgeAddition paper. Clarify that 3 is an inefficient algorithm for ensuring 3.2 based on an incomplete understanding of EdgeAddition algo

?? displays a partial alignment between two sequences. The continuous line represents two sites already aligned while the dotted line connects the site pair that is to be added. Updating the predecessor frontier for sequence S_1 would result in the following updates to $pred_A[x, i]$:

It is evident that once the position of x is greater or equal than that of an already aligned position, the alignment of a and b has no effect on the predecessor frontiers $pred_A[x, 2]$ for sites x with a position that is greater or equal than 3.
This property allows an alternative solution

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

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 frontier $succ$
Data: Predecessor frontier $pred$
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
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.
The member `sequences` of the `Closure` struct is implemented as a vector of structs containing vectors, again introducing overhead by pointer indirection.

`alig_set: Matrix<usize>` is a 2-dimensional matrix implementation containing **`usize`** elements, which are usigned integers with a size equal to the target architecture pointer size (meaning 64 bits on a 64 bit target).

Differences to Gabios-Lib

- matrices are contiguous memory instead of pointer of pointers -> reduces indirection and improves cache locality
- no unnecessary left and right buffers which need to be written each iteration
- frontier ops instead of pos matrix allows applying frontier changes in $O(\#changes)$ instead of $O(\#seq^2 * \text{part in while loop, don't know upper bound})$
- sequences is struct of matrices instead of Vec of struct of vecs -> less indirection

6 | Evaluation

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, 15]. 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 5 reference sets with different characteristics representing distinctive multiple alignment problems.
It is divided into:

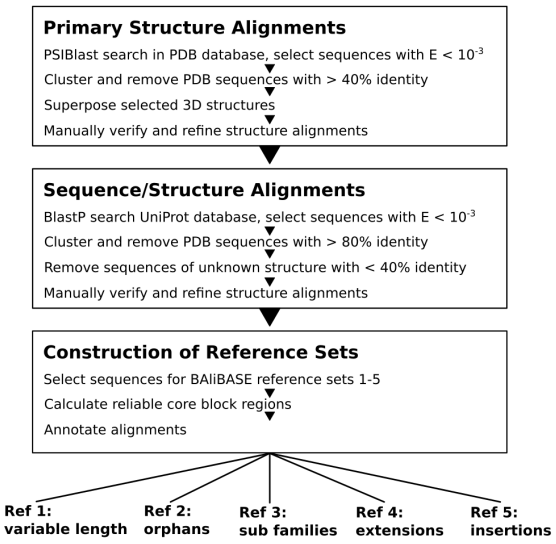

Figure 6.1: Semi automatic process used to establish the reference sets. Source: [1]

- reference set 1 subset V1, for which any two sequences share <20% identity and no internal insertions over 35 residues long
- 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 difficult 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* [15].

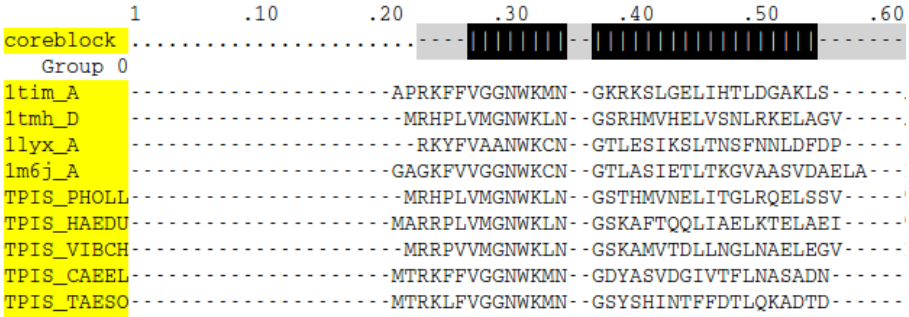

Figure 6.2: BAliBASE web interface. Black columns indicate core blocks. lbgi.fr

6.1.2 Quality of Alignments

Quality measures for protein alignment benchmarks
https://academic.oup.com/nar/article/38/7/2145/3100529

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. [16].

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 than that provided by the authors of BAliBASE [16] 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 part of the original implementation or not part of this version, it is not dependent on the *Expat XML parser*, which proved to be burdensome.
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

[1] View rustup.rs for installation instruction

cloned repository root. For usage instructions view `bali-score -h`.

6.4 Evaluated programs

6.4.1 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 [17].

- mafft offers plethora of strategies for different situations
- two strategies are evaluated
  - here referred to as Mafft-Fast: FFT-NS-1 (very fast; recommended for >2000 sequences; progressive method with a rough guide tree): mafft –retree 1 –maxiterate 0 input [> output]
  - here referred to as Mafft-Accurate: L-INS-i (probably most accurate; recommended for <200 sequences; iterative refinement method incorporating local pairwise alignment information): mafft –localpair –maxiterate 1000 input [> output]

MAFFT is evaluated for two different alignment strategies, from now on referred to as *fast* and *accurate*

6.4.2 Dialign

6.4.3 Spam-Align

- which pattern sets have been used? -> include rasbhari parameters

6.5 Results

- execution times include finding of micro alignments which is unnecessarily complex atm. and takes up roughly 50% of program run time

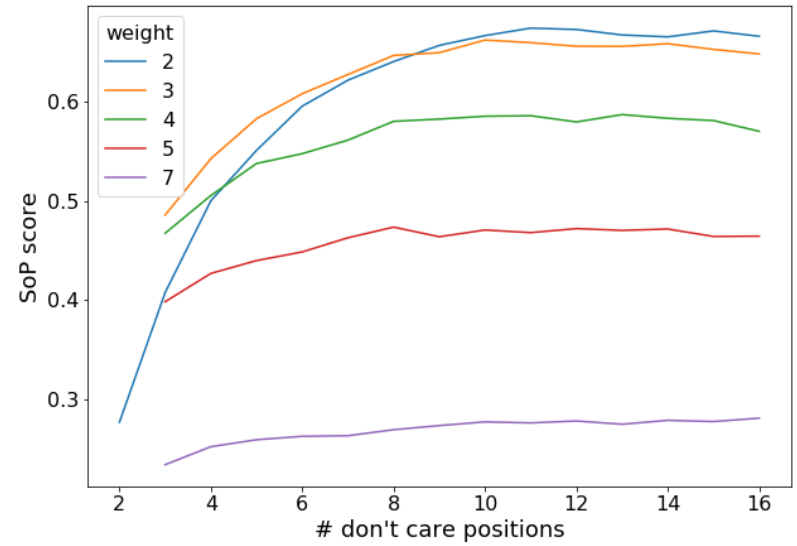

Figure 6.3: Sum of pairs scores aggregated over BAliBASE reference sets and weights.

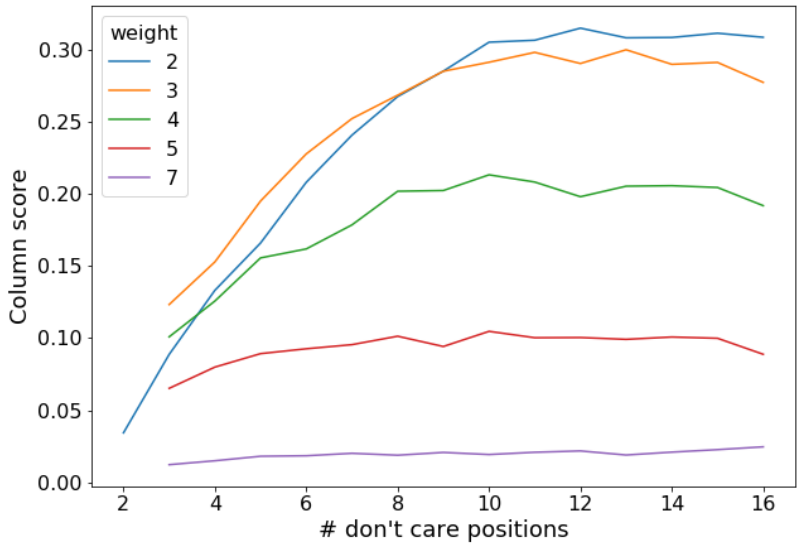

Figure 6.4: Column scores aggregated over BAliBASE reference sets and weights.

Table shows sum-of-pairs scores for compared programs (only selected parameters for spam-align).

	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-w-2_d-11	0.238148	0.718407	0.847409	0.713624	0.782116	0.726905
spam-align-w-3_d-10	0.222217	0.688189	0.840199	0.712141	0.778949	0.716480

3

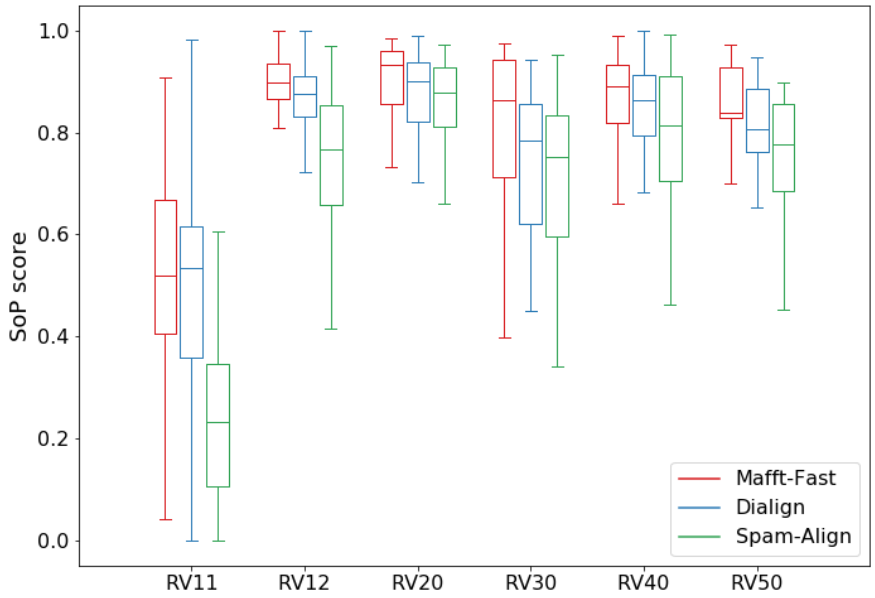

Figure 6.5: Sum of pairs scores. Spam-Align is called with a pattern of weight 2 and 11 don't care positions.

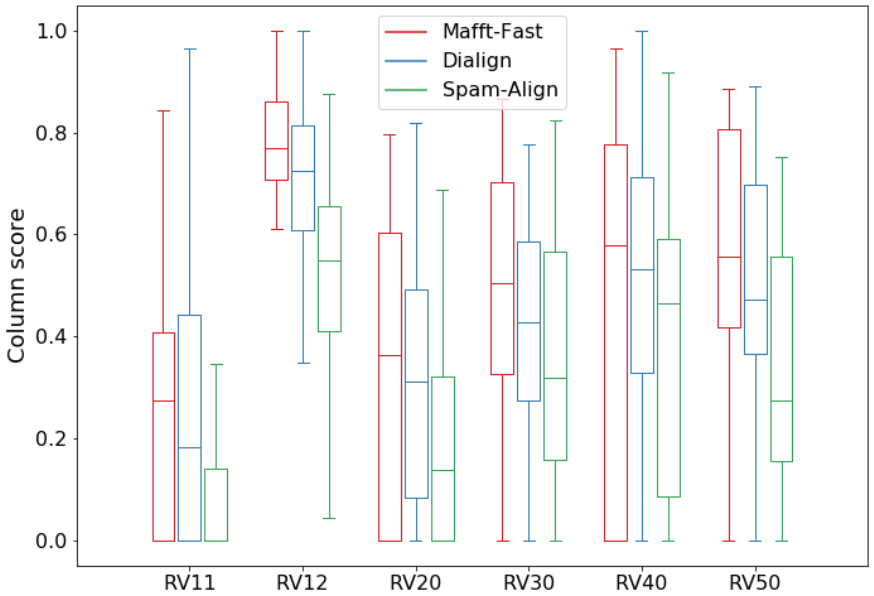

Figure 6.6: Column scores. Spam-Align is called with a pattern of weight 2 and 11 don't care positions.

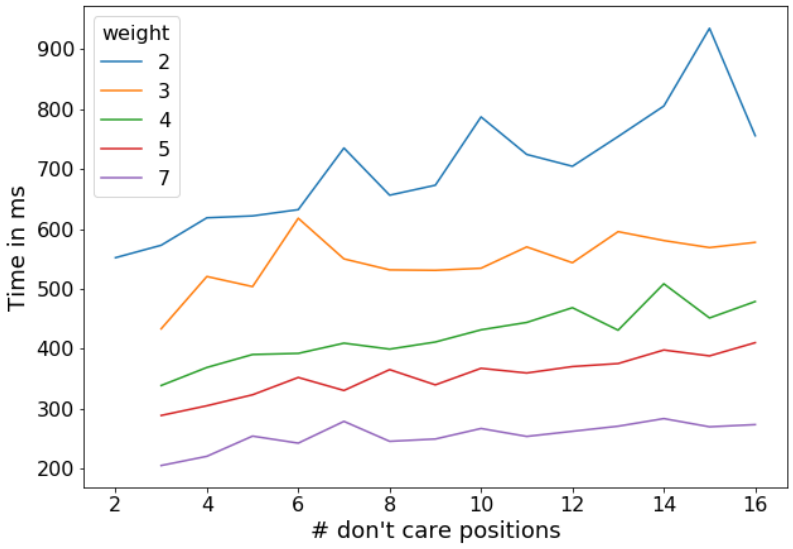

Figure 6.7: Mean execution times in ms.

7 Conclusion

7.1 Further work

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

7.1.1 Parallelisation Opportunities

Bibliography

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

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

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.