Segment-based Multiple Sequence Alignment

Dissertation
der Fakultät für Informations- und Kognitionswissenschaften
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Inform. Dipl.-Math. Amarendran Ramaswami Subramanian
aus Stuttgart

Tübingen
2009

Tag der mündlichen Qualifikation:	11.02.2009
Dekan:	Prof. Dr. Oliver Kohlbacher
1. Berichterstatter:	Prof. Dr. Michael Kaufmann
2. Berichterstatter:	Prof. Dr. Burkhard Morgenstern, Univ. Göttingen

This work is dedicated to my family.

ABSTRACT

In this PhD thesis the segment-based approach for multiple sequence alignment, initially introduced by the DIALIGN program, is thorougly investigated and substiantially improved. The segment-based approach belongs to the class of local alignment methods and thus is very strong in finding locally conserved motifs, whereas global methods align the input sequences globally from the beginning to end without specifically looking at locally occuring conserved motifs. Local alignments and especially segment-based methods therefore play an important role in molecular biology research, which is underscored by the fact that the results of this PhD thesis have already been extensively used in various biological research areas.

Initially we present a complete re-implementation of the DIALIGN approach in chapter 3 – DIALIGN-T – which also embraces several improvements, such as the exclusion of low-scoring sub-fragments and weight score factors yielding a statistical superior method on local and global benchmark databases. However DIALIGN-T still uses a greedy and, therefore, very naive strategy to build the final alignment so that in chapter 4 we re-formulate the assembling phase as an optimization problem that is NP-complete, but for which we can proove it to be a fixed parameter tractable (FPT) in the number of input sequences, under reasonable assumptions. Since we are interested in approaches that are useful in practice, we develop a plane-sweep algorithm that optimally solves the assembling problem whereby its computational time basically grows with the number of simultaneously occuring conflicting situations. By exploiting the ideas of the plane-sweep algorithm, we extend it, in chapter 5, to a full algorithmic framework which acts as a basis for developing further optimal or near-optimal heuristics for assembling an alignment from a given set of input similarities. Inspired by this framework, we improve, in chapter 6, DIALIGN-T to its most recent version DIALIGN-TX, which incorporates substiantial improvements by combining greedy and progressive strategies for assembling the alignment. In order to measure the quality of our improvements, we used the standard benchmark databases BAliBASE and BRaliBASE II on global alignments and the artifically generated databases IRMBASE and DIRMBASE on local alignments. The results show that DIALIGN-TX is currently outperforming all other methods on the local benchmark databases while still providing very good results on global alignments, i.e. it even outperforms the very popular global alignment program CLUSTAL W on the global benchmark database BAliBASE 3.

Altogether we conclude that DIALIGN-TX is one of the strongest methods on the important class of local alignments while still providing very good results on global alignments and consuming in practice only a reasonable amount of

computational time. In combination with the algorithmic framework we obtain a rich basis or future improvements to the segment-based approach for computing general and (biological) domain-specific multiple sequence alignments.

ZUSAMMENFASSUNG

In dieser Dissertation wird der Segment-basierte Ansatz zur Lösung des Multiplen Sequenz Alignment Problems, der initial mit dem DIALIGN Program eingeführt wurde, untersucht und substantiell weiterentwickelt. Der Segment-basierte Ansatz zählt zu den lokalen Alignment Methoden, die insbesondere bei der Suche nach lokal konservierten Motiven den globalen Methoden, die die Eingabesequenzen ohne spezifische Berücksichtigung von lokal konservierten Motiven global von Anfang bis Ende alignieren, überlegen sind. Lokalen Alignment Methoden und insbesondere der Segment-basierte Ansatz spielen aus diesem Grund eine wichtige Rolle in der molekularbiologischen Forschung. Dies zeigt sich unter anderem auch darin, dass die Ergebnisse dieser Arbeit bereits mehrfach für verschiedene biologische Fragestellungen erfolgreich eingesetzt wurden.

In Kapitel 3 wird mit DIALIGN-T eine signifikant verbesserte Re-Implementation des Segment-basierten Ansatzes vorgestellt. Im Rahmen dieser Verbesserungen werden insbesondere niedrig bewertete und damit störende Sub-Fragmente ausgeschlossen sowie zusätzlich über Gewichtungsfaktoren die Priorität der einzelnen Fragmente in der Assemblierungsphase optimaler vergeben. Jedoch wird innerhalb DIALIGN-T nach wie vor ein naives 'greedy' Verfahren eingesetzt, um das finale Alignment aus der Menge der errechneten Fragmente zusammenzufügen, so dass wir diese Assemblierungsphase daher in Kapitel 4 als mathematisches Optimierungsproblem formulieren, welches NP-vollständig ist. Wir zeigen jedoch, dass dieses Problem, unter angemessenen Rahmenbedingungen, 'Fixed Parameter Tractable' (FPT) in der Anzahl der Eingabesequenzen ist. Da wir uns für in der Praxis gut anwendbare Ansätze interessieren, entwickeln wir den sogenannten Plane-Sweep Algorithmus, der das Assemblierungsproblem exakt löst und dessen Komplexität im Wesentlichen nur mit der Anzahl der parallel auftretenden Konfliktsituationen steigt. Basierend auf der Grundidee des Plane-Sweep Algorithmus, leiten wir anschließend ein ganzes algorithmisches Frameworks in Kapitel 5 ab, welches als Grundlage zur systematischen Entwicklung weiterer optimaler und fast-optimaler Algorithmen/Heuristiken dient. Insbesondere wurde durch dieses Framework die Entwicklung von DIALIGN-TX in Kapitel 6 inspiriert. DIALIGN-TX stellt momentan die neueste Version des DIALIGN Ansatzes dar und setzt eine kombinierte Methode aus progressiven und greedy Strategien ein. Um die Alignment-Qualität zu messen, wurden für globale Alignments die Datenbanken BAliBASE 3 und BRAliBase II als Referenz verwendet; für lokale Alignments wurden die künstlichen Datenbanken IRMBASE und DIRMBASE erzeugt, die aus in Zufallssequenzen implantierte konservierten Motive bestehen. Anhand unserer Benchmark-Ergebnisse zeigen wir, dass DIALIGN-TX allen anderen aktuell populären Methoden auf lokalen Alignments qualitativ überlegen ist, aber auch zu sehr guten Resultaten auf

globalen Alignments führt. Insbesondere sind die Ergebnisse von DIALIGN-TX auf der globalen Benchmarkdatenbank BAliBASE 3 signifikant besser als die des sehr populären globalen Alignment-Programs CLUSTAL W.

Zusammenfassend schließen wir, dass DIALIGN-TX eine der stärksten Methoden auf der wichtigen Klasse der lokalen Alignments darstellt, dabei ebenfalls auf globalen Alignments sehr gute Ergebnisse liefert und im praktischen Einsatz innerhalb vernünftiger Zeitschranken läuft. In Kombination mit dem algorithmischen Framework erhalten wir eine reichhaltige Basis für zukünftige Verbesserungen des Segment-basierten Ansatzes zur Berechnung von allgemeinen und (biologisch) Domänen spezifischen Multiplen Sequenz Alignments.

CONTENTS

1. INTRODUCTION

In the more than fifty years ago since the DNA double-helix structure was explored by Watson and Crick [86], molecular biology has come a long way. With the years, the field of bioinformatics emerged to support various research areas in molecular biology by giving computational assistance in processing large input data sets with respect to biological questions, like 'Why are some organisms prone to a certain disease and others not?', and 'Is this species in an immediate or indirect ancestral relationship to another?'.

Within this PhD thesis we will focus on one of the basic problems in bionformatics: the *multiple sequence alignment* problem. We will especially discuss the segment-based approach to this problem. Briefly, when computing a multiple sequence alignment, one tries to find all the similarities that all, or just a part, of the given input protein or DNA/RNA sequences share. By solving that problem, biologists are supported in answering basic research questions, e.g. a gene is suspected to encode whether an organism is immune to a specific virus and biologists compare the genes of different organisms using a multiple sequence alignment to prove or disprove such a conjecture.

As already mentioned in [75], traditional approaches to multiple sequence alignment are either *global* or *local* methods. Global methods align sequences from the beginning to the end [14, 77, 34]. Based on the Nedleman-Wunsch objective function [59], these algorithms define the *score* of an alignment by adding up scores of *individual* residue pairs and by imposing *gap penalties*, they try to find an alignment with maximum total score in the sense of this definition. By contrast, most local methods try to find one or several conserved motifs shared by *all* of the input sequences [85, 47, 17]. During the last several years, a number of hybrid methods have been developed that combine global and local alignment features [53, 60, 8, 25]. One of these methods is the *segment-based* approach to multiple alignment [53], where alignments are composed of pairwise local sequence similarities. Altogether, these similarities may cover the entire input sequences – in which case a global alignment is produced – but they may as well be restricted to local motifs if no global homology is detectable. Thus, this approach can return global or local alignments – or a combination of both – depending on the extent of similarity among the input sequences.

Instead of comparing single residue pairs, the segment-based approach compares entire *substrings* of the input sequences to each other. The basic building blocks for pairwise and multiple alignment are un-gapped pairwise local alignments of each pair of the input sequences. Such local alignments are called *fragment alignments* or *fragments*; they may have any length up to a certain maximum length M. Thus, a fragment f corresponds to a *pair of equal-length substrings* of two of the input sequences. Pairwise or multiple alignments are composed of

such fragments; the algorithm constructs a suitable collection A of fragments that is *consistent* in the sense that all fragments from A can be represented *simultaneously* in one output multiple alignment.

Note that since multiple alignments are composed of local *pairwise* alignments, conserved motifs are not required to involve *all* of the input sequences. Unlike standard algorithms for local multiple alignment, the segment-based approach is therefore able to detect homologies shared by only two of the aligned sequences. With its capability to deal with both globally and locally related sequence sets and its ability to detect local similarities involving only a *subset* of the input sequences, the segment approach is far more flexible than standard methods for multiple alignment. It can be applied to sequence families that are not alignable by those standard methods; this is the main advantage of segment-based alignment compared to more traditional alignment algorithms.

This thesis is divided into four chapters, starting with the basic background material in chapter 2. After reviewing the basics and presenting a short overview of the presently popular alignment algorithms, we will introduce DIALIGN-T [75] in chapter 3 as a complete re-implementation of the segment-based approach based on the idea of DIALIGN [53]. DIALIGN-T already incorporates several algorithmic improvements that make DIALIGN-T statistically superior compared to the previous version DIALIGN 2.2 on various benchmark databases like the well-known BAliBASE 2.1 [78]. DIALIGN-T collects all fragments coming from a pairwise alignment phase and then adds them in a greedy way to the final multiple sequence alignment, i.e. all fragments are sorted in descending order by a specific weighting function and then added one by one into the alignment whereby fragments that do not fit into the alignment are discarded partially or even in full.

Assembling the alignment from a given set of fragments in a greedy way is quite a naive approach so we will investigate this sub-problem in more detail in chapter 4, where we show that it is NP-complete but luckily also fixed-parameter tractable in the number of sequences, under reasonable assumptions. As we head towards approaches that are useful in practice, we will then present a plane-sweep algorithm for optimally solving the assembling problem. The plane-sweep algorithm is only sensitive to conflicting situations that occur in parallel and therefore its computational complexity only grows with 'difficult' and distantly related input sequences. Inspired by this plane-sweep algorithm, we extend its idea to a full algorithmic framework in chapter 5, which forms a basis for systematically deriving optimal or near-optimal heuristics to solve the problem of assembling an alignment of a given set of input fragments while still being practically useful.

Once we have established this algorithmic framework, we will use it in chapter 6 to derive the DIALIGN-TX [74] program, which embraces a substantially improved method for assembling the final alignment from the fragments found in the pairwise alignment phase using a combined greedy and progressive strategy. DIALIGN-TX is based on a guide tree and to detect possible spurious sequence similarities, it employs a vertex-cover approximation on a conflict graph. We performed benchmarking tests on a large set of nucleic acid and protein sequences and conclude that for finding isolated homologous regions (i.e. the local alignment case) DIALIGN-TX statistically (significantly) outperforms all

other popular methods while still providing very good results on global alignments, which is underscored by the superiority of DIALIGN-TX compared to the very popular global aligner CLUSTAL W [40, 77] method on the global benchmark database BAliBASE 3.

Altogether, the relevance of the segment-based approach is supported by the extensive and successful use of the programs DIALIGN-T and DIALIGN-TX by the molecular biology community in various research areas [38, 30, 89, 61, 68, 2, 16, 67, 65] giving a good indication that the segment-based approach with its local character plays a relevant role in biological research. Finally, we present in chapter 7 further ideas about how to improve the segment-based approach, which are on the agenda for future work.

2. BACKGROUND

The problem of finding multiple sequence alignments (MSAs) arises from the fundamental biological motiviation of finding similarities and differences in a set of DNA, RNA or protein sequences in order to answer questions of phylogeny such as those concerning potential ancestral relationships or to facilitate clinical research. In this chapter, we outline the necessary biological background and explain the core problem from an algorithmic viewpoint as an optimization problem. Furthermore, we discuss the computational complexity and the classical algorithmic approaches of pairwise and multiple sequence alignments. For further background beyond what is outlined in this chapter the reader may consult the books from Gusfield [37] and Böckenhauer/Bongartz [7], upon this chapter is also built. Since finding multiple sequence alignments is an NP-complete problem, many heuristical algorithms and their implementations have evolved over the last two decades so at the end of this chapter we will give a short overview of some of the most popular current alignment programs.

2.1 Biological background

Deoxyribonucleic acid (DNA) is the basic building block for any presently known living organism by coding the genetic instructions of the constituent cells. Briefly, it is a recipe for the construction and functioning of each cell and thus for the overall organism by determining all relevant parameters such as the organism's size, shape, metabolism and its proneness to diseases. Therefore investigating the DNA of organisms is a fundamental issue in biological and clinical research. From a chemical point of view, the DNA consists of two opposite polymers that are comprised of the four nucleotides adenine (abbreviated by A), cytosine (C), guanine (G) and thymine (T) attached to a phosphate/sugar backbone. The nucleotides adenine and guanine are fused five- and six-membered heterocyclic compounds and belong to the so-called purines while cytosine and thymine are six-membered ring compounds that belong to the class of pyrimidines [5]. The two opposite polymers are arranged as a double helix [86] and are connected via hydrogen bonds between the attached nucleotides whereby they only exist between adenine and thymine and between guanine and cytosine. DNA sequences encode the information necessary for constructing proteins in specific regions that we call *genes*. Between these genes the non-coding regions are generally of unknown functionality although some are known to be involved in the regulation of the use of genes or general replication mechanisms.

Proteins play a vital role in almost all biological processes such as the construcion of cells and they are comprised of sequences of amino acids that are

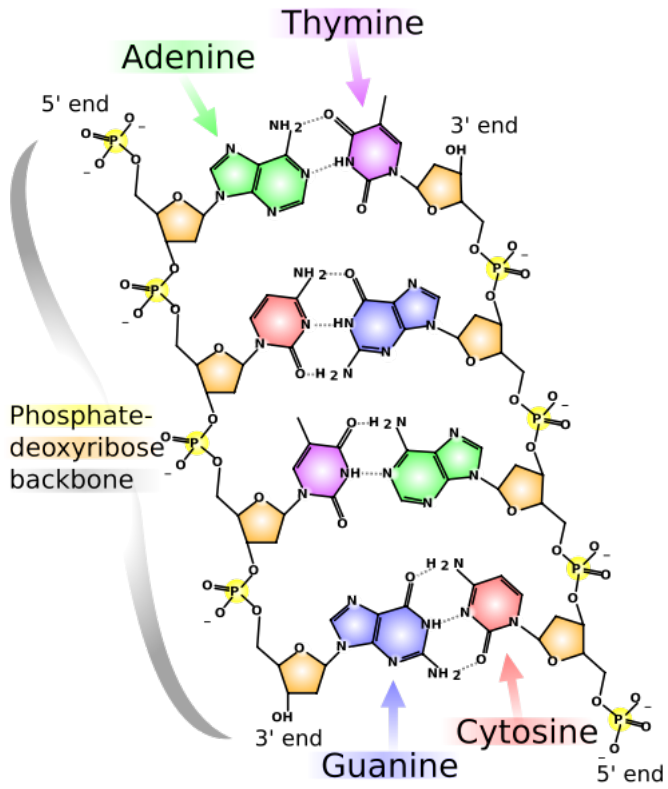

Fig. 2.1: Schematic depiction of the chemical DNA structure (courtesy of Madeleine Price Ball).

determined by their respective genetic coding region in the DNA. Amino acids consist of a central carbon denoted by C_α which has an amine (NH_2) (except proline) and a carboxyl ($COOH$) functional group attached and are distinguished by their individual organic substituent R (see Figure 2.2). Altogether there are 20 different standard amino acids occurring in proteins which are all listed in Table 2.1 together with their abbreviations and their codons that encode them in the DNA. Apart from the 20 standard amino acids there exist a few very rare "non-standard" amino acids that seldom occur in proteins, like selenocysteine [22].

Proteins emerge by the chaining of amino acids along peptide bonds between the carboxyl and the amine group. The sequence of characters representing the chain of amino acids of a protein is referred to as the *primary structure* of a protein. The secondary 3D-structure of a protein also plays a very crucial role in molecular biology and bioinformatics (e.g. [28]), however, its consideration is omitted in the discussion of the multiple sequence alignment problem within this thesis.

The process of constructing proteins from genes as coding regions in the DNA is comprised of two steps, transcription and translation. During the first transcription step, the DNA is copied into the so-called messenger RNA (mRNA)

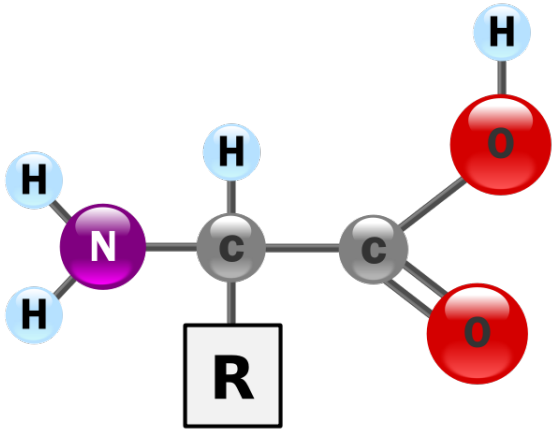

Fig. 2.2: The general structure of an amino acid where the carboxyl group is located on the right and the amino group on the left (courtesy of Yassine Mrabet).

by RNA polymerase, which takes place in the cell nucleus. Subsequently, the mRNA is translated by ribosomes into the sequence of amino acids outside the cell nucleus in the cytoplasm. In this process, three consecutive nucleotides, also referred to as codons, always encode one amino acid. Combinatorially there are 64 possible codons that encode the twenty standard amino acids as well as the three stop codons that indicate the end of a protein coding region, which are the TAA, TGA and the TAG codons. The beginning of a coding region is regularly only indicated by codons that also encode amino acids, like ATG for methionine, which is also the predominant start codon in eukaryotes.

2.2 Problem description

DNA and protein sequences have been decoded for many organisms on a character level, i.e. the sequences and their nucleotides or amino acids are known and represented by strings of characters. However, biologists are interested in the investigation of functional or evolutionary homologies in a set of sequences, which can be translated into the problem of aligning the given sequences such that the similarities and differences can be clearly depicted. Yet, real-world data tends to be very large, which makes it impossible to work out the alignments manually and thus automated algorithms are required to compute the alignments within a reasonable time. In the following sections we formulate the general problem of finding multiple sequence alignments as an optimisation problem which is, unfortunately, NP-complete and thus very time-consuming to solve. As a result, many heuristics have been implemented over the last two decades and we will outline some of the popular ones including DIALIGN-T and DIALIGN-TX, which are subject to more detailed discussion in chapters 3 and 6.

Name	Abbrev.	Letter	Codons
Alanine	Ala	A	GCT GCC GCA GCG
Arginine	Arg	R	CGT CGC CGA CGG AGA AGG
Asparagine	Asn	N	AAT AAC
Aspartic acid	Asp	D	GAT GAC
Cysteine	Cys	C	TGT TGC
Glutamine	Gln	Q	CAA CAG
Glutamic acid	Glu	E	GAA GAG
Glycine	Gly	G	CGT GGC GGA GGG
Histidine	His	H	CAT CAC
Isoleucine	Ile	I	ATT ATC ATA
Leucine	L	Leu	TTA TTG CTT CTC CTA CTG
Lysine	K	Lys	AAA AAG
Methionine	Met	M	ATG
Phenylalanine	Phe	F	TTT TTC
Proline	Pro	P	CCT CCC CCA CCG
Serine	Ser	S	TCT TCC TCA TCG AGT AGC
Threonine	Thr	T	ATC ACC ACA ACG
Tryptophan	Trp	W	TGG
Tyrosine	Tyr	Y	TAT TAC
Valine	Val	V	GTT GTC GTA GTG

Tab. 2.1: Amino acids together with their abbreviations and their codons.

2.2.1 Computing pairwise alignments

Before we formally explain the problem of computing multiple sequence alignments, we will start with the sub-problem of computing pairwise alignments, which is basically a problem of matching two strings that encode a DNA or protein sequence by their alphabet; for DNA sequences, the alphabet is defined by the nucleotides as

$$\Sigma_D = \{A, C, G, T\}$$

and for protein sequences by the amino acids as

$$\Sigma_P = \{A, V, L, I, F, P, M, S, T, C, W, Y, N, Q, D, E, K, R, H, G\}.$$

An alignment of two strings, then, is defined by inserting gaps in the two strings such that equal or 'similar' letters occur at the same position. More precisely, the formal definition of an alignment is as follows:

Definition 2.1
Let $t = t_1 \dots t_m$ and $u = u_1 \dots u_n$ be two strings of an alphabet Σ, i.e. $t, u \in \Sigma^*$ whereby the gap symbol $-$ is not part of Σ. Let $\Sigma' = \Sigma \cup \{-\}$ the gap-amended alphabet and $h : (\Sigma')^* \to \Sigma^*$ a homomorphism defined by $h(a) = a$ for all $a \in \Sigma$ and $h(-) = \lambda$, whereby λ denotes the empty string.

Then a (pairwise) alignment of t and u is defined as a pair of strings (t', u') each of length $l \geq \max\{m, n\}$ over the alphabet Σ' such that the following conditions hold:

1. $l = |t'| = |u'| \geq \max\{|t|, |u|\}$ ($|\cdot|$ denotes the length of a string),

2. $h(t') = t$ and $h(u') = u$,

3. there is no position i at which t' and u' both have a gap represented by the gap symbol $-$, i.e. for all $1 \leq i \leq l$ either $t_i \neq -$ or $u_i \neq i$ is true.

For example let $t =$ `GACGGATTATG` and $u =$ `GATCGGAATAG` be two DNA sequences. Then a possible alignment of t and u could be

$$t' = \texttt{GA-CGGATTATG}$$
$$u' = \texttt{GATCGGAATA-G}$$

which has one gap in each sequence and only one mismatch at the 8th position. In this example, the alignment seems to be 'good' because a lot of positions contain equal characters in both sequences. The quality assessment of alignments plays a central role in finding biologically meaningful alignments, which brings us to the definition of scoring systems that assess quality of alignments.

Definition 2.2
A scoring system s is given as a function that assigns a real-valued score to an alignment as follows:

$$s : \Sigma'^* \times \Sigma'^* \to \mathbb{R}.$$

Let t and u be two sequences and s a scoring system. Then an alignment (t', u') of t and u is called optimum with respect to s if it maximizes the score $s(t', u')$ among all possible alignments of t and u.

Of course, the complexity of algorithms for finding optimum alignments greatly depends on the underlying scoring system. In the next subsection we will discuss the most popular scoring systems that provide a good trade-off between biological quality of the optimum alignments with respect to the scoring system and the resulting algorithmic complexity.

Scoring systems and substitution matrices

Almost all popular scoring systems are based on a symmetric *substitution matrix*

$$w : \Sigma \times \Sigma \to \mathbb{R}$$

that is derived from the mutation probabilities and is used together with a gap penalty $g \in \mathbb{R}$ that penalizes insertions and deletions. The scoring for two characters $a, b \in \Sigma'$ is then $s(a, b) = w(a, b)$ if $a, b \in \Sigma$ and set to $s(a, b) = g$ if a or b equals the gap character $-$. Eventually the score for an alignment $s(t', u')$ of two sequences t and u is defined by the sum of s along all positions

$$s(t', u') := \sum_{1 \leq i \leq |t'| = |u'|} s(t'_i, u'_i) = \sum_{1 \leq i \leq |t'| = |u'|} w(t'_i, u'_i)$$

We call such a scoring system *simple* and we will shortly see that finding an optimum alignment with an underlying simple scoring system is quite easy using dynamic programming approaches. Sometimes the simple scoring system is

extended by using affine gap penalties that impose higher costs for the opening of a gap compared to costs for the extension of a gap. More formally, it is comprised of a substitution matrix $w = (\cdot, \cdot)$, an gap-open penalty $g_o \in \mathbb{R}$ and a gap-extension penalty $g_e \in \mathbb{R}$ and for an alignment (t', u') of two sequences $t, u \in \Sigma'^*$ the score $s(t'_i, u'_i)$ for any position is defined by

$$s(t'_i, u'_i) = \begin{cases} w(t'_i, u'_i) & \text{if } t'_i, u'_i \neq - \\ g_o & \text{if } t'_i = - \text{ or } u'_i = - \text{ and } i = 1 \\ g_o & \text{if } i > 1 \text{ and } t'_i = - \text{ and } t'_{i-1} \neq - \\ g_o & \text{if } i > 1 \text{ and } u'_i = - \text{ and } u'_{i-1} \neq - \\ g_e & \text{else.} \end{cases}$$

In other words, a fragment of consecutive gap characters in one sequence of the alignment is scored by $g_o + lg_e$ whereby l is the length of this segment full of gaps. This has the effect that opening a gap is more costly than extending an already existing one; g_o and g_e are mostly chosen to be negative. Such a *simple scoring system with affine gap costs* is often used for locally related sequences, which means that they are expected of having homologies only in certain local areas and are not expected to be globally homologous.

However, in order to obtain a biologically relevant alignment, we are very much dependent on the choice of a suitable substitution matrix. Usually, for DNA sequences one chooses $m \geq 1$ and a match is scored by m, a mismatch by $-m$ and the gaps are penalized by $g < 0$ whereby the latter varies across the different algorithms but usually we have $g << -1$ since insertions or deletions in DNA sequences are less likely to occur compared to mutations whereby all 12 different mutations occur with equal probability. For protein sequences, the mutation probabilities between the different amino acids vary due to the chemical structure and thus a simple 'match and mismatch scoring' as in the DNA case would be too coarse-grained. As a result, various scoring matrices for amino acids have been developed. For amino acid substitution there are two commonly used series of matrices: the PAM (Point Accepted Mutation) matrices, that were developed in the late seventies [15] and the very popular BLOSUM (BLOcks of amino acid SUbstitution Matrix) [39] matrices.

The PAM substitution matrix:
We will now explain how the PAM matrices are constructed. In order to do that, we need to have a notion of *an accepted mutation*, which is defined to be a mutation that is not impairing the functionality of the overall protein and thus it can be successfully inherited. Let t and u be two protein sequences; then we can say that t and u have a *PAM-distance* of 1 if t can be constructed from u by a series of accepted point mutations (i.e. no deletions and insertions) such that there is one accepted point mutation in every hundred amino acids on average. We can easily extend the definition of this distance measure for distances > 1 and we observe that two protein sequences that have a PAM-distance of k do not necessarily differ in k percent of the positions since multiple mutations may occur at the same position. Now, a k-PAM-matrix is a substitution matrix that is suitable for comparing two protein sequences that have a PAM-distance of k but we will first explain how we would construct it under ideal circumstances. We assume that we have arbitrarily many pairs of homologous protein sequences of equal biological functionality and of PAM-distance of k for each of those pairs.

Furthermore, we assume that we know the optimum alignment for each of these pairs and we call the set of these pairs of aligned sequences A. Now we can define the symmetric k-PAM-matrix PAM_k as

$$PAM_k(a_i, a_j) = \log\left(\frac{freq(a_i, a_j)}{freq(a_i) \cdot freq(a_j)}\right), a_i, a_j \in \Sigma_P,$$

whereby $0 \leq freq(a_i, a_j) \leq 1$ is the relative occurrence of an aligned column containing a_i and a_j in A and $0 \leq freq(a_i) \leq 1$ is the relative occurrence of a_i in all sequences of A. Basically the PAM matrix entry for a_i and a_j describes the relative probability of the occurrence of (a_i, a_j) as accepted mutation in an alignment versus a random occurrence of this pair. Of course we only described how the PAM matrix would be constructed on the basis of a given ideal set of homologous sequence pairs of PAM-distance k, which is usually not the case. In practice, however, we start with a set A of pairs of very similar sequences of a common ancestor and that have a PAM-distance of one. Due to their high similarity it is very easy to determine the best biological alignment for each sequence pair, from which we can compute the PAM_1 matrix and $freq(\cdot,\cdot), freq(\cdot)$ as above. Let F be a (20×20)-matrix such that the entry

$$F(i, j) = \frac{freq(a_i, a_j)}{freq(a_i)}$$

describes the mutation probability $a_i \rightarrow a_j$ for each pair of amino acids (a_i, a_j) in the set A (independently of the occurrence of a_i). The kth power F^k of F gives us an approximation of the mutation probability $a_i \rightarrow a_j$ for each pair (a_i, a_j) of amino acids in sequences of PAM-distance k which allows us to approximate the k-PAM matrix as follows:

$$PAM_k(i, j) = \log\left(\frac{freq(a_i) \cdot F^k(i, j)}{freq(a_i) \cdot freq(a_j)}\right) = \log\left(\frac{(F^k(i, j)}{freq(a_j)}\right).$$

Lemma 2.1
The above definition of PAM_k yields a symmetric matrix.

Proof: We will show the symmetry of PAM_k by induction over k. By the definition of F, this is obvious for $k = 1$. Assuming that PAM_{k-1} is symmetric, we then get the following equation for an arbitrary pair (a_i, a_j) of amino acids:

$$\begin{aligned}
PAM_k(i, j) &= \log\left(\sum_{1 \leq l \leq 20} \frac{F^{k-1}(a_i, a_l) \cdot freq(a_l, a_j)}{freq(a_l) \cdot freq(a_j)}\right) \\
&= \log\left(\sum_{1 \leq l \leq 20} \frac{F^{k-1}(a_l, a_i) \cdot freq(a_l, a_j)}{freq(a_i) \cdot freq(a_j)}\right) \\
&= \log\left(\sum_{1 \leq l \leq 20} \frac{F^{k-1}(a_l, a_i) \cdot freq(a_j, a_l)}{freq(a_i) \cdot freq(a_j)}\right) \\
&= PAM_k(j, i).
\end{aligned}$$

The second equation is due to the symmetry of PAM_{k-1} and the third is due to the symmetry of $freq(\cdot,\cdot)$ which completes the proof.

Usually the PAM-distance of two given input sequences from which an alignment should be computed is unknown, therefore in practice, PAM-matrices with the standard values of $k = 40, 100, 250$ are used.

The BLOSUM substitution matrix:
Another very popular series of substitution matrices are the BLOSUM matrices [39], which are built from the BLOCKS-database which contains information about homologous regions of closely related proteins that have been retrieved from a set of multiple sequence alignments. We will shortly outline the construction of the BLOSUM matrices starting with looking at gap-free intervals in the above mentioned multiple sequence alignments that we call *blocks*. The BLOCKS-database is comprised of blocks that have a certain minimum length. A block in the BLOCKS-database having length n and that is is part of a multiple sequence alignment of k sequences $s_1, \ldots, s_k$ can be regarded as an $n \times k$ matrix $B(i, j)$. Some rows of such a block can be identical or almost identical, however, we are now interested in those pairs of lines that do not coincide in at least a certain percentage of positions. More formally, we regard all pairs of matrix entries $(B(i_1, j), B(i_2, j)), 1 \leq j \leq n$ such that the rows i_1 and i_2 differ in at least x percent of positions. In order to determine the BLOSUM-x-matrix, we compute the set P of row-pairs of all blocks in the BLOCKS-database that differ in at least x percent positions. We then derive the relative occurrences $freq(a_i)$ and $freq(a_i, a_j)$ for each pair of amino acids (a_i, a_j) in this set P. Then the BLOSUM-x-matrix is then given by

$$BLOSUM_x(i, j) = 2 \cdot \log_2 \left(\frac{freq(a_i, a_j)}{freq(a_i) freq(a_j)} \right), a_i, a_j \in \Sigma_P$$

whereby in practice these values are rounded to the next integer. In contrary to the PAM-matrices, which are computed from very closely related sequence pairs, the BLOSUM-matrices are derived from evolutionarily distantly related protein sequences (at least for high x), which is of advantage since computing an alignment becomes less trivial for distantly related input sequences. The BLOSUM62 matrix (see Figure 2.3) in this series turned out to be the most valuable representative because it produces the best results in various algorithmic setups for multiple sequence alignments [77, 75]) and, therefore, is also used in the DIALIGN-T and DIALIGN-TX algorithms.

Global pairwise alignments

In this subsection we will elaborate on the problem of finding an optimum global pairwise alignment of two DNA or protein sequences with respect to the underlying scoring scheme. Global alignments focus on aligning the sequences completely in an optimal way whereas local alignments, which are the subject of the next subsection, aim at finding homologous substrings of the given input sequences. We start with the definition of the optimization problem.

Definition 2.3
Let $\Sigma = \Sigma_D$ or $\Sigma = \Sigma_P$ be an alphabet and

$$s : \Sigma'^* \times \Sigma'^* \rightarrow \mathbb{R}$$

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Fig. 2.3: The BLOSUM62 substitution matrix for amino acids ([39]).

be a scoring system on Σ'. Then the optimization problem of two input sequences $t, u \in \Sigma^*$ is given by finding an alignment $(t', u') \in \Sigma'^* \times \Sigma'^*$ such that the score $s(t', u')$ is maximized.

Initially, we will discuss an algorithm for computing optimum pairwise global alignments in the case of a simple scoring system being comprised of a substitution matrix w and a gap penalty $g < 0$ based on a dynamic programming approach which is a special case of the Needleman-Wunsch algorithm [59]. Thereafter, we will present a modified algorithm for the case of a simple scoring system with affine gap costs often also referred to as the Gotoh-algorithm [31].

Hence, let $t = t_1 \dots t_m$ and $u = u_1 \dots u_n$ be two sequences over the alphabet Σ and s be a simple scoring system with the substitution matrix w and the gap penalty g. Including the empty word, we have $m+1$ prefix strings of t and $n+1$ prefix strings of u, on which we iteratively compute a $(m+1) \times (n+1)$-matrix M containing the similarity scores of the optimum alignment for each pair of prefixes, i.e. $M(i, j)$ is constructed such that it contains the optimum alignment of $t_1 \dots t_i$ and $u_1 \dots u_j$. Of course, $M(m, n)$ then contains the overall optimum alignment of t and u. In the following we will call M the *similarity matrix* of t and u. The basic idea of the dynamic programming approach is to compute the optimum alignment of prefixes of t and u from the optimum alignment of shorter prefixes that have been determined in a previous step of the algorithm. Initially, we fill the first row $M(i, 0), 1 \leq i \leq m$ and the first column $M(0, j), 1 \leq j \leq n$ with the trivial alignment of the prefix aligned with the respective number of gap symbols and, consequently, we get the scores

$$s(M(i, 0)) = g \cdot i$$

and

$$s(M(0, j)) = g \cdot j.$$

Now we want to explain how to obtain the optimum alignment between $t_1 \dots t_j$ and $u_1 \dots u_i$ to be stored in $M(i, j)$ under the assumption that we know the

best alignment for all shorter prefixes of t and u, i.e. all prefix pairs of t and u with maximum length $i-1$ and j or i and $j-1$. Looking at a pairwise alignment as a $(2 \times l)$-matrix we have three possibilities for how the last column in the optimum alignment for $t_1 \dots t_j$ and $u_1 \dots u_i$ may look. Either the last column is given by (t_i, u_j) (case 1) or $(t_i, -)$ (case 2) or $(-, u_j)$ (case 3). Since s is a simple scoring scheme that sums up the sums scores for each column, we conclude for the three cases as follows:

- Case 1 (match/mismatch): $M(i,j)$ is given by $M(i-1, j-1)$ amended by the column (t_i, u_j) having score $s(M(i-1, j-1)) + w(t_i, u_j)$.
- Case 2 (insertion): $M(i,j)$ is given by $M(i-1, j)$ amended by the column $(t_i, -)$ having score $s(M(i-1, j)) + g$.
- Case 3 (deletion): $M(i,j)$ is given by $M(i, j-1)$ amended by the column $(-, u_j)$ having score $s(M(i, j-1)) + g$.

Therefore, we go through all three cases and choose the one that maximizes the score $s(M(i,j))$ to determine $M(i,j)$. If we iterate from row $M(i, \cdot)$ to row $M(i+1, \cdot)$ and the columns from left to right we only have to store the relevant case in $M(i,j)$ and a pointer to its respective predecessor entry $M(i-1, j-1)$, $M(i, j-1)$ or $M(i, j-1)$. Altogether we obtain the following theorem.

Theorem 2.1
The Needleman-Wunsch algorithm described above computes an optimum global pairwise alignment of two sequences $t = t_1 \dots t_n$ and $u = u_1 \dots u_m$ with respect to a simple scoring scheme s in linear running time $O(nm)$ and space $O(nm)$.

The simple scoring scheme penalizes each gap with a unique gap cost g whereas it often makes more sense biologically to impose lower costs to an extension of an already existing gap compared to the costs of opening a new gap, for example, when comparing complementary DNA (cDNA) with genome DNA. Let us remember the above mentioned affine gap costs that are given by a gap-open penalty g_o and a gap-extension penalty g_e and so a gap of length l is then penalized by

$$g(l) := g_o + l \cdot g_e.$$

Having a simple scoring scheme with affine gap costs still allows us to use a dynamic programming approach to compute an optimum alignment. The algorithm we will outline in the following is also referred to as the Gotoh algorithm [31]. Again let $t = t_1 \dots t_m$ and $u = u_1 \dots u_n$ be the sequences over the alphabet Σ we want to align using a simple scoring scheme with affine gap costs comprised of a substitution matrix w, a gap open penalty $g_o \in \mathbb{R}$ and a gap extension penalty $g_e \in \mathbb{R}$. As before, we iteratively compute the similarity matrix $M(\cdot, \cdot)$ that contains the best known alignment for the prefixes $t_1 \dots t_i$ and $u_1 \dots u_j$ in $M(i,j)$. We additionally introduce the matrices $T(\cdot, \cdot)$ and $U(\cdot, \cdot)$ such that $T(i,j)$ (respectively $U(i,j)$) contains the best alignment for the prefixes $t_1 \dots t_i$ and $u_1 \dots u_j$ ending with a gap in t (respectively u). The matrices are initialized as follows

- $M(0,0)$ is set to the empty alignment with score 0

- $U(0, j)$ is set to the empty alignment with an artificial score of $-\infty$ for all $1 \leq j \leq n$.
- $M(i, 0) = U(i, 0)$ is set to the trivial alignment (i.e. i-th prefix of t aligned against i gap symbols) with score $g(i)$ for all $1 \leq i \leq m$.
- $T(i, 0)$ is set to the empty alignment with an artificial score of $-\infty$ for all $1 \leq i \leq m$.
- $M(0, j) = T(0, j)$ is set to the trivial alignment (i.e. j-th prefix of u aligned against j gap symbols) with score $g(j)$ for all $1 \leq j \leq n$.

Again we iterate row by row through the matrix index by the prefix lengths and explain how to compute the optimum alignment for (i, j) in $M(i, j), T(i, j), U(i, j)$ according to their definition under the assumption that we already have computed the optimum alignment for shorter prefixes of t and u in M, T and U, i.e. prefix pairs of length maximum $i - 1$ and j or i and $j - 1$. First we update the auxiliary matrices T and U beginning with $T(i, j)$, for which we have the cases of extending an already existing gap ending in t (Case 1) or opening a new gap (Case 2), i.e.

- Case 1: $T(i, j-1)$ amended by the column $(-, u_j)$ having score $s(T(i, j-1)) + g_e$
- Case 2: $M(i, j-1)$ amended by the column $(-, u_j)$ having a score of at least $s(M(i, j-1)) + g_o + g_e$

Depending on which case yields the higher score, we update $T(i, j)$. Analogously, we obtain the following two similar cases for U and store the better one in $U(i, j)$:

- Case 1: $U(i-1, j)$ amended by the column $(t_i, -)$ having score $s(U(i-1, j)) + g_e$
- Case 2: $M(i-1, j)$ amended by the column $(t_i, -)$ having a score of at least $s(M(i-1, j)) + g_o + g_e$

Having computed $T(i, j)$ and $U(i, j)$ we are now equipped with all prerequisites to determine an optimum solution for prefixes of length i and j of t and u using affine gap costs to be stored in $M(i, j)$. According to the case of non-affine gap costs we end up with the follwing three cases for $M(i, j)$:

- Case 1: $M(i-1, j-1)$ amended by the column (t_i, u_j) having score $s(M(i-1, j-1)) + w(t_i, u_j)$
- Case 2: $T(i, j)$ having a score of at least $s(T(i, j))$
- Case 3: $U(i, j)$ having a score of at least $s(U(i, j))$

Depending which case yields the highest score, we set $M(i, j)$ which concludes the description of the algorithm. Again we apply the same strategy of storing the alignments in T, U and M as in the case of non-affine gap costs and state the following theorem:

Theorem 2.2
The Gotoh algorithm described above computes an optimum pairwise global alignment of two sequences $t = t_1 \dots t_n$ and $u = u_1 \dots u_m$ with respect to a simple scoring scheme s with affine gap costs in linear running time $O(nm)$ and space $O(nm)$.

Finally it is important to know that the quadratic space requirement $O(nm)$ in both algorithms presented in this subsection can be reduced to linear space using the divide and conquer strategy of the Hirschberg algorithm [41].

Local pairwise alignments

In contrast to a global pairwise alignment, a local alignment of two input sequences t and u is a global alignment of appropriate substrings of t and u. Finding optimum local pairwise alignments becomes biologically relevant when one is interested in homologous regions of two input sequences that are highly divergent outside these regions. More precisely, the problem is defined as follows:

Definition 2.4
Let t and u be two sequences over an alphabet Σ ($\Sigma = \Sigma_P$ or $\Sigma = \Sigma_D$) and let s be a simple scoring scheme with the substitution matrix w and gap penalty $g < 0$. Then an optimum local alignment of t and u is given by an optimum pairwise alignment (t'_l, u'_l) of two substrings t_l and u_l of t and u such that its score $s(t'_l, u'_l)$ is maximized over all such substrings of t and u.

Similar to the case of global pairwise alignments, we apply a dynamic programming approach , which is also called the Smith-Waterman algorithm [71] and it is pretty much similar to the Needleman-Wunsch algorithm. We store the optimum alignment of a suffix pair for the prefixes $t_i, \dots, t_j$ and $u_1, \dots, u_j$ of t and u in the similarity matrix M under $M(i, j)$. Since we are interested in optimum alignment of the best substrings of t and u, we initially set $M(i, 0) = 0$ and $M(0, j) = 0$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$. We now explain how to compute $M(i, j)$ under the assumption that we have computed the entries of M for all shorter prefixes of t and u, i.e. all prefix pairs of t and u with maximum length i and $j - 1$ or $i - 1$ and j. Contrary to the Needleman-Wunsch algorithm, we have now to deal with four instead of three cases in order to determine $M(i, j)$:

- Case 1: $M(i, j)$ is set to the the empty suffix pair and thus has score 0.
- Case 2: $M(i, j)$ is given by $M(i-1, j-1)$ amended by the column (t_i, u_j) having score $s(M(i-1, j-1)) + w(t_i, u_j)$.
- Case 3: $M(i, j)$ is given by $M(i-1, j)$ amended by the column $(t_i, -)$ having score $s(M(i-1, j)) + g$.
- Case 4: $M(i, j)$ is given by $M(i, j-1)$ amended by the column $(-, u_j)$ having score $s(M(i, j-1)) + g$.

Again we set $M(i, j)$ to the case that maximizes the score. In contrast to the Needleman-Wunsch algorithm for global pairwise alignments, the optimum local alignment is not necessarily stored in $M(n, m)$ since $M(n, m)$ contains the optimum global alignment for the best suffix pair of t and u. The optimum local alignment can, however, be found in the overall matrix M at the position with maximum score. Applying the same strategy for storing the alignments in M as in the Needleman-Wunsch algorithm we obtain the following theorem:

Theorem 2.3
The Smith-Waterman algorithm described above computes an optimum local pairwise alignment of two sequences $t = t_1 \dots t_n$ and $u = u_1 \dots u_m$ with respect to a simple scoring scheme s in linear running time $O(nm)$ and space $O(nm)$.

Also, in this case, the quadratic space requirement $O(nm)$ can be reduced to a linear space consumption using the divide and conquer strategy of the Hirschberg algorithm [41].

Heuristics for database searches

Although the algorithms presented in the previous subsection provide the exact optimum pairwise alignments in polynomial time, they are often not satisfactory for searches in large DNA or protein databases since they require too much computational time. As a result, often faster but heuristical algorithms that do not guarantee an optimum solution but a satisfying approximation thereof within reasonable computing time are used in practice. Among the various heuristics, the most popular ones are the FASTA [62] and the BLAST [4] methods. Both methods are based on finding very short substrings of the query sequence q and all database sequences $d \in D$ of exact or very high scoring similarity and subsequently try to assemble them to longer alignments by taking only the candidates of high statistical significance into account, whereby the statistical significance is assessed by the probability that such an alignment may occur at random.

2.2.2 The general multiple sequence alignment problem

After having elaborated the computation of pairwise sequence alignments, we now discuss the problem of optimally aligning a number of input sequences $t_1, \dots, t_k$ over an alphabet Σ according to a scoring scheme suitable for assessing the quality of multiple sequence alignments.

Definition 2.5
Let $t_1 = t_{11} \dots t_{1m_1}, \dots, t_k = t_{k1} \dots t_{km_k}$ be k sequences over an alphabet Σ (Σ=Σ_D or Σ=Σ_P), i.e. $t_i \in \Sigma^*, 1 \leq i \leq k$ whereby the gap symbol $-$ is not part of Σ. Let $\Sigma' = \Sigma \cup \{-\}$ be the gap-amended alphabet and $h : (\Sigma')^* \to \Sigma^*$ a homomorphism defined by $h(a) = a$ for all $a \in \Sigma$ and $h(-) = \lambda$, whereby λ denotes the empty string.

Then a multiple sequence alignment of $t_1, \dots, t_k$ is defined as a tuple of strings $(t'_1, \dots, t'_k)$ of length $l \geq \max\{m_i | 1 \leq i \leq k\}$ over the alphabet Σ' such that all of the following conditions hold:

1. $l := |t'_1| = |t'_2| = \ldots = |t'_k|$,
2. $h(t'_i) = t_i$ for all $1 \leq i \leq k$,
3. there is no position $1 \leq j \leq l$ at which all t'_i have a gap represented by the gap symbol $-$, i.e. for all $1 \leq j \leq l$ there exists a $1 \leq i \leq k$ where $t'_{ij} \neq -$.

The length $l = |t'_i|$ is also called the length of the multiple sequence alignment $(t_1, \ldots, t_k)$

In the following we will focus on a simple scoring scheme for assessing the biological relevance of a multiple sequence, which is a straightforward extension of the simple scoring scheme for pairwise alignments by summing up the pairwise alignment scores.

Definition 2.6
A general scoring scheme s for multiple sequence alignments over an alphabet Σ is given by a function

$$s : (\Sigma'^*)^* \to \mathbb{R}.$$

A sum of pair scoring scheme, or shortly SP-scoring scheme, for multiple sequence alignments over an alphabet Σ is induced by a symmetric substitution matrix

$$w(\cdot, \cdot) : \Sigma' \times \Sigma' \to \mathbb{R}$$

over the gap amended alphabet $\Sigma' = \Sigma \cup \{-\}$ as follows: the SP-scoring s on multiple sequence alignments of k sequences is given by the function $s : (\Sigma'^*)^k \to \mathbb{R}$ whereby

$$s(t_1, \ldots, t_k) = \sum_{1 \leq i < j \leq k} w(t_i, t_j).$$

Having an SP-scoring s for multiple sequence alignments over an alphabet Σ, we can now define the problem of finding an optimum multiple sequence alignment.

Definition 2.7
Let $t_1 = t_{11} \ldots t_{1m_1}, \ldots, t_k = t_{k1} \ldots t_{km_k}$ be k sequences over an alphabet Σ and let s be a scoring scheme for multiple sequence alignments. Then a multiple sequence alignment $(t'_1, \ldots, t'_k)$ is called optimum with respect to s if $s(t'_1, \ldots, t'_k)$ is maximized on the set of all possible multiple sequence alignments of $(t_1, \ldots, t_k)$. We define the associated decision problem SP-MS-Align as follows:
Input: The input is given by an integer $k > 0$ and a set of sequences $T = (t_1, \ldots, t_k)$ over an alphabet Σ together with an SP-scoring scheme s over Σ and a threshold value M.
Output: YES, if there is a multiple sequence alignment of T with an s-score equal or greater than M. Otherwise the output is NO.

In contrast to finding optimum pairwise alignments, the problem of finding an optimum multiple sequence alignment is far more difficult, as it can be shown [84] to be NP-complete even in the case of a simple underlying SP-scoring.

s	0	1	a	b	$-$
0	-2	-2	-1	-2	-1
1	-2	-2	-2	-1	-1
a	-1	-2	0	$-k$	-1
b	-2	-1	$-k$	0	-1
$-$	-1	-1	-1	-1	0

Tab. 2.2: SP-scoring scheme for the construction in the proof of theorem 2.4

Theorem 2.4
The decision problem SP-MS-Align is NP-complete.

Proof [84, 7]: Initially we look at the Dec(0,1)-Shortest-Superseq (Dec(0,1) stands for decimals 0 and 1) problem which is given by the question of whether a given set $S = (s_1, \ldots, s_k)$ of strings over the alphabet $\{0, 1\}$ have a common supersequence of length $|t| \leq N$, whereby N is a threshold as part of the input. We remember that a supersequence of $S = (s_1, \ldots, s_k)$ is a string r such that for all $1 \leq i \leq k$ s_i can be obtained from r by deleting characters. Middendorf [50] has already shown that Dec(0,1)-Shortest-Superseq is NP-complete so we are now going to give a polynomial time reduction of Dec(0,1)-Shortest-Superseq to SP-MS-Align in order to prove the theorem.

Therefore, let $S = (s_1, \ldots, s_k)$ be a set of strings over the alphabet $\{0, 1\}$ and N be the threshold value of the given Dec(0,1)-Shortest-Superseq instance. We now construct $N + 1$ SP-MS-Align instances by using the same sequences amended by two sequences using the new symbols A and B such that the alphabet for the multiple sequence alignment problem is given by $\Sigma = \{0, 1, a, b\}$. Furthermore, we will now define the SP-scoring s and the threshold M of the SP-MS-Align instance such that every solution of the multiple sequence alignment with a score $\geq M$ contains no column with 0 and 1. Obviously, such a solution would yield a supersequence for S so that we then only have to show that it has a maximum length of N. Since we do not know how many 0- and 1-columns we are going to end up with, we have to construct $N+1$ instances of the SP-MS-Align problem as follows:

- For all $i, j \in \mathbb{N}$ with $i + j = N$, we define $X_{i,j} = S \cup \{a^i, b^j\}$
- We set the threshold value $M := -((k-1) \cdot \|S\| + (2k+1) \cdot N)$, whereby $\|S\| = \sum_{1 \leq i \leq k} |s_i|$ is defined as the total length of all strings in S.
- Let the SP-scoring scheme s be defined as in Table 2.2.

It is now sufficient to show that S has a common supersequence r of length $\leq N$ if and only if one of the $X_{i,j}$ admits a multiple sequence alignment with SP-score $\geq M$ with respect to s.

For the first implication, we assume that we have for one $X_{i,j}$ a multiple sequence alignment $A = (s'_1, \ldots, s'_k, \alpha, \beta)$ with score $s(A) \leq M$, whereby α denotes the a-row and β the b-row. We define A' to be the restriction of A to the first k rows of $s'_1, \ldots, s'_k$ and will now prove that the score of A' is always $(k-1) \cdot \|S\|$,

regardless of A. Let us look at the single columns of A' where we observe that a column with l gap symbols has a scoring of

$$-(l \cdot (k-l) + 2 \cdot \frac{(k-l)(k-l-1)}{2}) = -(k-1) \cdot (k-l).$$

If x denotes the number of columns in A' and y the number of gap symbols in A' we get $\|S\| = k \cdot x - y$. Furthermore, let l_p denote the number of gaps in column p of A' for $1 \leq p \leq x$; then we obtain for the scoring of A'

$$\begin{aligned} s(A') &= -\sum_{p=1}^{x} ((k-1) \cdot (k-l_p)) = -(k-1) \sum_{p=1}^{x} (k-l_p) \\ &= -(k-1) \cdot (k \cdot x - y) = -(k-1) \cdot \|S\|. \end{aligned}$$

In the next step we are going to prove that $X_{i,j}$ admits an alignment of SP-score $\geq M$ only if such an alignment has length $\leq N$ and it has no column where a and 1 or b and 0 occur in parallel.

In the case when $x \geq N$, the comparison of the rows α and β reduces the score by at least N if there is no column in which a and b appear simultaneously. Every column in which a and b both occur reduces this score further by at least k. In the other case where we have $x < N$, there are overlaps of a and b symbols in the rows α and β and the SP-score is therefore maximum $-(N + (N-x) \cdot k)$.

Now we determine the SP-score of the comparisons of α and β with $s'_1, \ldots, s'_k$. Let x again be the length of the alignment A and z the number of pairs of gap symbols in this comparison which then has a score of at most $-2k \cdot x + z$. Since α contains $x - i$ and β contains $x - j$ gap symbols, we conclude, together with the fact that $i + j = N$, that $z \leq k \cdot (x - N)$. Hence the comparison of α and β with $s'_1, \ldots, s'_k$ contributes an SP-score of at most

$$-2kx + k \cdot (x - N) = -k \cdot (x + N).$$

Of course, the columns in which a occurs together with 1, or b occurs together with 0, reduce the SP-score further. Altogether we conclude for the overall score for the alignment that

$$s(A) \leq -(k-1) \cdot \|S\| - \max\{N, N + k(N-x)\} - k(x+N). \tag{2.1}$$

The right side of inequality 2.1 is maximized if $x \leq N$. If there is no column in which a and 1 or b and 0 occur in parallel, the right side is exactly M, which we have defined to be

$$M = -((k-1) \cdot \|S\| + (2k+1) \cdot N).$$

Thus, if there is an alignment of $X_{i,j}$ with SP-score $\geq M$, then it must have length $\leq N$ and there is no column that contains a and 1 or b and 0. From the latter we immediatly see that there is also no column where 0 and 1 occur in parallel and thus the common supersequence $r = r_1 \ldots r_N$ of $s_1, \ldots, s_k$ of length N can be as follows: For each $1 \leq l \leq N$ we set $r_l = 0$ if $\alpha_l = a$ and $r_l = 1$ if $\beta_l = b$.

Now it remains to prove that if we are given a common supersequence r of $s_1, \ldots, s_k$ of length N we find an $X_{i,j}$ that admits a multiple sequence alignment of score $\geq M$. Let i be the number of 0-symbols and j be the number of 1-symbols in r. It is sufficient to show that there exists a multiple sequence alignment of $X_{i,j}$ with an SP-score of at least M. Since r is a supersequence, there is an alignment between r and each s_i without mismatches for all $1 \leq i \leq k$. To construct a multiple sequence alignment of $X_{i,j}$, we take all these mismatch-free alignments and align the A symbols with the 0-columns and the B-symbols with the 1-columns. Analogous to the calculations for inequality 2.1, we find that this alignment has score M, which finishes the proof of the theorem.

Recalling the Needleman-Wunsch algorithm described in subsection 2.2.1 and its dynamic programming approach, we are now going to extend it for the computation of optimum multiple sequence alignments with respect to an SP-scoring. Therefore, let $T = (t_1, \ldots, t_k)$ be a set of $k > 0$ input sequences over an alphabet Σ and s an associated SP-scoring scheme. Let n be the maximum length of all sequences in T. Because we want to extend the two-dimensional dynamic programming approach to k dimensions , we construct a k-dimensional similarity matrix $M(\cdot, \ldots, \cdot)$. For any $1 \leq i_j \leq |t_j|$ with $1 \leq j \leq k$ we store the optimum alignment for the prefixes $t_{11} \ldots t_{1i_1}, \ldots, t_{k1} \ldots t_{ki_k}$ in $M(i_1, \ldots, i_k)$, similar to the two-dimensional problem, and of course M has $O(n^k)$ entries. Additionally, we fix a lexicographical ordering on the k-dimensional index of M induced by the sequence number in first order and the prefix length in second order. The traversal order for computing M is then along this lexicographical ordering. In the algorithm we initially set $M(0, \ldots, 0)$ to the empty alignment with score 0 and we set $M(0, \ldots, i_j, \ldots, 0)$ for all $1 \leq j \leq k$ and all $1 \leq i_j \leq |t_j|$ to the trivial alignment of the i_j-th prefix of t_j aligned with i_j number of gap symbols in every other sequence $\neq t_j$, which has a score of

$$s(M(0, \ldots, i_j, \ldots, 0)) = i_j \cdot \frac{k(k-1)}{2} w(-,-) + (k-1) \sum_{1 \leq l \leq i_j} w(t_{jl}, -).$$

Let $0 \leq i_j \leq |t_j|$ for $1 \leq j \leq k$ and assume we already know the optimum alignment for all prefixes

$$t_{11} \ldots t_{1(i_1 - d_1)}, \ldots, t_{k1} \ldots t_{k(i_k - d_k)}$$

with $(d_1, \ldots, d_k) \in \{0,1\}^k \backslash \{(0, \ldots, 0)\}$ in $M(i_1 - d_1, \ldots, i_k - d_k))$ so that we again have to show how we compute the optimum alignment for the prefixes $t_{11} \ldots t_{1i_1}, \ldots, t_{1k} \ldots t_{ki_k}$ in $M(i_1, \ldots, i_k)$ therefrom. Since we have a k-dimensional similarity matrix M, we have to consider $2^k - 1$ cases which are naturally enumerated by all tuples $(d_1, \ldots, d_k) \in \{0,1\}^k \backslash \{(0, \ldots, 0)\}$. For each case $(d_1, \ldots, d_k)$, the candidate alignment for $M(i_1, \ldots, i_k)$ consists of $M(i_1 - d_1, \ldots, i_k - d_k)$ amended by a column $c = (c_1, \ldots, c_k) \in \Sigma'^k$ that has for each sequence $t_j, 1 \leq j \leq k$ a gap symbol if $d_j = 0$ and t_{ji_j} if $d_j = 1$. For each of these cases the SP-score is given by

$$M(i_1 - d_1, \ldots, i_k - d_k) + \sum_{1 \leq a < b \leq k} w(c_a, c_b).$$

After having computed all these $2^k - 1$ cases, we choose the one with the highest score and store it in $M(i_1, \ldots, i_k)$ and finally end up with the optimum align-

ment in $M(|t_1|, \ldots, |t_k|)$. Again similar to the pairwise case, we do not want to waste space and therefore actually only store the winning case in $M(i_1, \ldots, i_k)$ together with a pointer to the predecessor entry $M(i_1 - d_1, \ldots, i_k - d_k)$.

Theorem 2.5
Let $k > 0$ and $t_1, \ldots, t_k$ be k sequences over the alphabet Σ together with an SP-scoring s. Then the above described extension of the Needleman-Wunsch algorithm computes an optimum multiple sequence alignment in time $O(2^k n^k)$ whereby n denotes the length of the longest sequence.

Due to its exponential time bound, the dynamic programming approach for finding optimum multiple sequence alignments is only suitable for input instances comprised of very few and very short sequences thus significantly restricting its usefulness in practice. As a result, many approximative heuristics have emerged and we will give an overview of the presently most popular ones in section 2.3. However, before that, we will have a closer look at the segment-based approach, which is the central topic of this thesis.

2.2.3 The segment-based approach

As already described in [75], traditional approaches to multiple sequence alignment are either *global* or *local* methods. Global methods align sequences from the beginning to the end [14, 77, 34]. Based on the Needleman-Wunsch objective function [59], these algorithms define the *score* of an alignment by adding up scores of *individual* residue pairs and by imposing *gap penalties*; they try to find an alignment with maximum total score in the sense of this definition. By contrast, most local methods try to find one or several conserved motifs shared by *all* of the input sequences [85, 47, 17].

During the last several years, a number of hybrid methods have been developed that combine global and local alignment features [53, 60, 8, 25]. One of these methods is the *segment-based* approach to multiple alignment in the DIALIGN family [53, 75, 74] where alignments are composed of pairwise local sequence similarities. Altogether, these similarities may cover all input sequences – in which case a global alignment is produced – but they may also be restricted to local motifs, occurring only in some of the input sequences, if no global homology is detectable. Thus, this approach can return global or local alignments – or a combination of both – depending on the extent of similarity among the input sequences.

However, since multiple alignments are composed of local *pairwise* alignments, conserved motifs are not required to involve *all* of the input sequences. Unlike standard algorithms for local multiple alignment that only detect conserved motifs shared by all input sequences, the segment-based approach is therefore additionally able to detect homologies shared by only two of the aligned sequences. With its capability to deal with both globally and locally related sequence sets and with its ability to detect local similarities involving only a *subset* of the input sequences, the segment approach is far more flexible than standard methods for multiple alignment. It can be applied to sequence families that are not alignable by those standard methods; this is the main advantage of segment-based alignment compared to more traditional alignment algorithms.

Both local and global approaches are useful for different classes of inputs coming from various biological domains and the segment-based methods like the DIALIGN family turned out to perform especially well on local alignments while still being a well-performing alternative when seeking global alignments. We will elaborate on this further in chapters 3 and 6.

We will now explain the idea of the segment-based approach in more detail. As already mentioned, instead of comparing single residue pairs, the segment-based approach compares entire *substrings* of the input sequences to each other. The basic building blocks for pairwise and multiple alignment are un-gapped pairwise local alignments involving two of the sequences under consideration. Such local alignments are called *fragment alignments* or *fragments*; they may have any length up to a certain maximum length M. Thus, a fragment f corresponds to a *pair of equal-length substrings* of two of the input sequences, or more precisely:

Definition 2.8
Let $t_1, \ldots, t_k \in \Sigma^*$ be k sequences over an alphabet Σ. A segment of sequence t_i is a triple $(t_i, p_i, l_i) \in \Sigma^l$ with $p_i, l_i > 0$ and $1 \leq p_i + l \leq |t_i|$ and naturally associated with

$$(t_i, p_i, l_i) = t_{ip_i} \ldots t_{i(p_i+l-1)}.$$

A fragment f between two sequences t_i and t_j is then a gap-free alignment of two segments (t_i, p_i, l) and (t_j, p_j, l) of equal length l and we denote it by $f = (t_i, p_i, t_j, p_j, l)$. Let s be an SP-scoring scheme over Σ; then the score of a fragment is naturally given by

$$s(f) = \sum_{0 \leq r < l} s(t_{i(p_i+r)}, t_{j(p_j+r)}).$$

In the segment-based approach, pairwise or multiple alignments are composed of such fragments and algorithms following this approach construct a suitable collection A of fragments that is *consistent* in the sense that all fragments from A can be represented *simultaneously* in one output multiple alignment.

2.3 Common multiple sequence alignment approaches

As already mentioned, global alignment approaches try to align the input sequences from beginning to the end whereas local methods are focused on finding locally conserved motifs that share only a part or even all input sequences. In the following subsection we will give a short overview of the currently most popular alignment programs for the two classes of global and local alignment approaches. In the chapters 3 and 6 especially, we will elaborate in more detail on their qualitative performance on various benchmarks of locally and globally related input sequences.

2.3.1 Global alignment strategies

CLUSTAL W

CLUSTAL W is a very widely used alignment program for computing predominantly global alignments [40, 77] and it implements a progressive approach

introduced in the 1980s for the classical multiple sequence alignment problem in [26].

CLUSTAL W basically consists of the three following steps. Initially, CLUSTAL W computes a distance matrix that contains a numerical score indicating the divergence for each pair of input sequences. Therefore, an approximative pairwise alignment is computed for each pair of input sequences based on k-tuple matches together with a fixed penalty for every gap. Typically, $k = 1, 2$ for protein and $k = 2, 3, 4$ for nucleotide sequences. Additionally, CLUSTAL W offers alternatively the option to use full dynamic programming for computing optimum pairwise alignments in this step using a simple scoring scheme with affine gap costs.

After having calculated the distance matrix, CLUSTAL W builds a guide tree from all input sequences using the neighbour-joining method [69]. This guide tree assumes a phylogenetic order of the input sequences based on the scores in the previously computed distance matrix.

The final stage progressively aligns larger and larger groups of sequences along the guide tree. Starting at the leaves, it aligns two groups of alignments identified by the child nodes of each parent node in a bottom-up manner. Thus, at each stage two sequences or sub-alignments are aligned using a full dynamic programming algorithm whereby the score between one position in one alignment/sequence and another is determined by the average of all SP-scores from the underlying scoring scheme. In the process of aligning two sub-alignments/sequences gaps are penalized by affine gap costs and gaps that have been previously introduced in one sub-alignment remain fixed.

T-COFFEE

T-COFFEE (Tree-based Consistency Objective Function For alignmEnt Evaluation) is quite similar to CLUSTAL W. However it incorporates some substantial improvements [60]. Similar to CLUSTAL W, it initially determines a distance matrix in order to obtain a divergence measurement for each pair of sequences. The major improvement comes with the computation of a library of alignment information using all pairwise alignments between each pair of sequences by using local and global alignment algorithms. This library also takes transitive information into account, i.e. the similarity scoring of segment s_1 in sequence t_1 and s_2 in sequence t_2 is also influenced by the similarity of segments s_3 in sequence t_3 with s_1 and s_2.

After having computed the library, the guide tree is determined using the neighbour-joining method and finally the alignment is computed along this guide tree making extensively use of the library. In general, T-COFFEE thus provides qualitatively higher output alignments compared to CLUSTAL W.

A recent extension of T-COFFEE is M-COFFEE [81] which is a meta-method of assembling multiple sequence alignments combining the output of several individual methods including those outlined in this subsection.

MUSCLE

The approach implemented in the alignment program MUSCLE (MUltiple Sequence Comparison by Log-Expectation) [25] is comprised of three stages. Similar to CLUSTAL W, a distance matrix is computed whereby a k-mer distance is used. A k-mer is a contiguous subsequence of length k. Related sequences tend to have more k-mers in common than expected by chance and thus the k-mer distance is derived from the fraction of k-mers in common in a compressed alphabet. Upon this distance matrix a guide tree is constructed using the UPGMA (Unweighted Pair Group Method with Arithmetic Mean) method [72]. A progressive alignment is constructed by following the branching order of this tree.

The second stage improves the progressive alignment by using the output of the previous stage to compute the Kimura distance [45] between each pair of sequences; the computation of the Kimura distance requires an underlying alignment. By the use of the Kimura distance, a finer distance matrix than the one of the previous step is computed which is then used to compute an improved multiple sequence alignment in subsequent progressive steps.

The final iterative refinement stage deletes an edge from the tree of stage 2 and performs a progressive alignment for the two sub-trees whereby the subtrees are arranged using a log-expectation scoring. Finally, the alignments of the two sub-trees are aligned to one final alignment. If the resulting alignment scores better (w.r.t. to an SP-scoring) it is kept as the currently best known, otherwise it is discarded. The last stage is repeated until convergence or a user defined-limit has been reached.

PROBCONS

PROBCONS implements a different approach [19] while still having a progressive stage. Altogether five stages take place in PROBCONS starting with the computation of the posteriori-probability matrix for the underlying Pair Hidden Markov Model for each pair of residues of different sequences. Thereafter, the so-called *probability consistency transformation* is computed, which additionally incorporates the similarity of residues x and y to residues of sequences (other than those containing x and y) into the xy pairwise comparison.

Thereafter, a guide tree is constructed using a scoring from the resulting alignment of the Hidden Markov Model of the first stage and a hierarchical clustering. The guide tree then forms the basis for carrying out a progressive alignment whereby the previously computed probability consistency transformation acts as underlying scoring and gaps are penalized by 0. Finally, and similar to MUSCLE, the alignment is partitioned randomly into two groups of sequences and then realigned whereby the better outcome is preserved.

MAFFT

The MAFFT multiple sequence alignment program offers a variety of different sub-methods with different accuracy and running times [43, 42]. The G-INS-i,

L-INS-i and E-INS-i methods especially produce the most qualitatively valuable output in general and are comprised of four steps. Initially, pairwise alignments are constructed whereby G-INS-i uses global alignment with an Fast Fourier Transformation (FFT) approximation [43], whereas the other two methods L-INS-i and E-INS-i incorporate local alignment information using the FASTA34 program [62]. From that a guide tree is constructed using the UPGMA approach with a modified linkage.

Then every pairwise alignment is divided into gap-free fragments. Those fragments then get assigned an importance value involving the SP-score and the frequency of its residues occurring in gap-free fragments. The importance values for each fragment are then further processed into an importance matrix assigning a score for each pair of residues of different sequences. Extending this importance matrix to sub-alignments, it is used to progressively build the multiple sequence alignment along the guide tree. In a final stage, the resulting alignment is iteratively refined based on the approaches described in [6, 32] using the weighting scheme proposed in [33]. We remark that the E-INS-i method also performs quite well on local alignments, which we will see in more detail in chapter 6.

2.3.2 Local alignment strategies

In contrast to global alignment strategies, which have been more emphasized in past research, the local alignment algorithms that focus more on finding locally isolated homologous regions have emerged recently and now also play a significant role in biological research. In the following we will outline the most popular representatives of this class, including DIALIGN-T and DIALIGN-TX, both of which have been developed as part of this thesis.

POA

The POA (Partial Order Alignment) [48, 35] starts with a very fast construction of a distance matrix for each pair of input sequences like the previously described global methods, but using the BLAST bit scores [4]. The BLAST bit score is, briefly, a value calculated from the number of gaps and substitutions associated with each aligned sequence - the higher the score, the more significant the alignment. Next a guide tree is constructed via the application of agglomerative nearest-neighbour clustering, i.e. Kruskal's Minimum Spanning Tree algorithm [13].

In this algorithm a multiple sequence is regarded as a DAG (directed acyclic graph), in which individual sequence letters are represented by nodes. Directed edges are drawn between consecutive letters in each sequence and aligned residues are fused into a single node whereby redundant edges are pruned. This representation of an alignment in the progressive stage along the guide tree is used when aligning two sub-alignments based on a Smith-Waterman scoring [71].

POA obtains its local character by using, on the one hand, a Smith-Waterman-based scoring and, on the other hand, a truncated gap penalization, which highly

favours few but very long gaps over many small gaps.

DIALIGN

The approach implemented in DIALIGN [53, 51] differs quite a lot from the previously described methods as it does not use any progressive strategy. Essentially, DIALIGN is a segment-based algorithm starting off at the initial stage by computing optimum segment-based pairwise alignments using an *objective weight function*. More precisely, a resulting pairwise alignment contains a set of *consistent* fragments, where 'consistency' means that they all can be realized in parallel in the alignment. Contrary to the other approaches that are based on an SP-scoring with a more or less complicated gap penalization, DIALIGN assigns a higher weight to a fragment the less probable its occurrence is at random (with at least the same SP-score in sequences of the same length). For that it uses a substitution matrix s (BLOSUM62 for protein sequences and simple match/mismatch for DNA matrices) and approximatively computes for each fragment f having length l_f between sequence s_1 of length l_1 and sequence s_2 of length l_2 the probability $P(s(f), l_f, l_1, l_2)$ of the random occurrence of a fragment in sequences of length l_1 and l_2 having the same length l_f and at least the similarity score $s(f)$. Finally, the weight score $w(f)$ is set to the negative logarithm of this probability, i.e

$$w(f) := -\log\left(P(s(f), l_f, l_1, l_2)\right).$$

Using this weight score, DIALIGN computes for each pair of input sequences the optimum set of consistent fragments accordingly, using a modified space-efficient dynamic programming approach. In its final stage, DIALIGN sorts all these fragments in descending order w.r.t the weight scoring $w(\cdot)$ and adds them greedily to a final multiple sequence alignment: beginning with the highest scoring it decides whether the current fragment fits into the already existing alignment (i.e. whether it is *consistent* with it) and if so, it adds it; otherwise the fragment will be discarded.

DIALIGN-T

DIALIGN-T is a complete re-implementation of DIALIGN which incorporates several improvements [75] yielding a significant advantage over several benchmark databases. In the pairwise stage, DIALIGN is restricted to fragments of length 40 whereas DIALIGN-T supports fragments up to a length of 100. However, this raises the problem of very long fragments containing low-scoring middle parts, where it would be more appropriate to break it into two or more sub-fragments. This is compensated for in DIALIGN-T by not allowing low scoring regions in long fragments thus ending up with only those very long fragments that actually carry biologically meaningful information from the beginning to the end. The two other major improvements take place in the modified greedy stage, where we use a modified weighting for determining the order in which the fragments are considered. In this modified weighting we scale the weight score $w(f)$ of each fragment f of sequences s_1 and s_2 by a factor coming from the overall similarity score of the sequences s_1 and s_2 in relation to the similarity

score of all other sequence pairs. This procedure favours fragments coming from highly similar sequences over fragments between less similar sequences. The second improvement in this stage is that fragments that are found not to fit into the alignment are not discarded as a whole but are cut such that the remainder becomes consistent and is re-inserted into the sorted list for later consideration. As part of this thesis we will describe DIALIGN-T in more detail in the next chapter.

DIALIGN-TX

Besides some newer features, such as the support of anchor points, DIALIGN-TX has been substantially improved by using a combination of greedy and progressive strategies in the stage where fragments from the pairwise alignment phase are combined. The new method we developed initially computes a guide tree T for the set of input sequences based on their pairwise similarity scores. We divide the set of fragments contained in the respective optimal pairwise alignments into two subsets F and G, where F consists of all fragments with weight scores below the average fragment weight score in all pairwise alignments, and G consists of the fragments with a weight score above or equal to the average weight. In the first step, the set G is used to calculate an initial multiple alignment A in a progressive manner along the guide tree T. During this progressive step, two sub-alignments are merged using a conflict graph that contains edges for every conflict between any pair of fragments of the two sub-alignments (to be merged). Of course there may be conflicts that involve triples or more generally m-tuples of fragments, however, we only take care of 'pair conflicts' since they have the highest probability of occurring. We resolve the pair conflicts by removing a vertex cover computed by the 2-approximation of Clarkson [11] for the weighted vertex cover problem. The remainder is then aligned greedily as implemented in the previous version of DIALIGN-T. The low-scoring fragments from set G are later added to A in a, again, greedy way after the progressive phase has been completed, provided they are consistent with A. In addition, we construct an alternative multiple alignment B using the greedy approach implemented in the previous version DIALIGN-T. Finally, the program returns either A or B, depending on which one of these two alignments has the highest weight score. In chapter 6 we will explain DIALIGN-TX in more detail.

3. THE DIALIGN-T PROGRAM

In this chapter, we present a complete re-implementation of the segment-based approach to multiple protein alignment that contains a number of improvements compared to the previous version *DIALIGN 2.2*. DIALIGN-T has been developed as part of this thesis and the results have already been published in [75], which is the basis for this chapter.

The DIALIGN approach is superior to Needleman-Wunsch-based multi-alignment programs on *locally* related sequence sets. However, it is often outperformed by these methods on datasets with *global* but weak similarity at the primary-sequence level.

In this chapter, we discuss the strengths and weaknesses of DIALIGN in view of the underlying *objective (weight) function*. Based on these results, we propose several heuristics to improve the segment-based alignment approach. For pairwise alignment, we implemented a fragment-chaining algorithm that favours chains of low-scoring local alignments over isolated high-scoring fragments. For multiple alignment, we use an improved *greedy* procedure that is less sensitive to spurious local sequence similarities. To evaluate our method on globally related protein families, we used the well-known database *BAliBASE* [78]. For benchmarking tests on locally related sequences, we created a new reference database called *IRMBASE*, which consists of simulated conserved motifs implanted into non-related random sequences.

On the BAliBASE, DIALIGN-T performs significantly better than the previous version of DIALIGN and is comparable to the standard global aligner CLUSTAL W, though it is outperformed by some recently developed programs that focus on global alignment. On the locally related test sets in IRMBASE, our method outperforms all other programs that we evaluated.

In the next section we will describe the detailed procedure for determining the fragments from the pairwise dynamic programming. The subsequent section describes the amended method of assembling those fragments such that the influence of less related sequences is reduced and a more sophisticated way of dealing with inconsistent fragments is used. Finally, we will compare DIALIGN-T with other popular alignment programs on benchmark databases and present the results before we end this chapter with a conclusion.

3.1 Pairwise computation of the fragments

In this section we describe the pairwise alignment phase of DIALIGN-T, where, for each pair of sequences, the optimum set of consistent fragments is determined

according to the carefully chosen object weighting function.

3.1.1 The objective function in DIALIGN-T

From a computer scientist's point of view, sequence alignment is an *optimisation problem*. Most alignment algorithms are – explicitly or implicitly – based on an *objective function*, i.e. on some kind of *scoring scheme* assigning a quality score to every possible alignment of a given input sequence set. Based on such a scoring scheme, different optimisation algorithms are used to find optimal or near-optimal alignments. For multiple alignment, a variety of optimisation techniques have been proposed. These algorithms differ substantially from each other with regard to their computational complexity and in view of their ability to find or approximate numerically optimal alignments. However, the most important feature of an alignment program is not the optimisation algorithm that it uses, but rather the underlying objective function that is used to score possible output alignments. If the objective function is *biologically wrong* by assigning high scores to biologically meaningless alignments, then even the most efficient optimisation algorithms are only efficient in finding mathematically high-scoring *nonsense* alignments. With a more realistic objective function, however, even simple-minded heuristics may lead to biologically plausible alignments.

The objective function that we use in the segment-based approach of DIALIGN-T is defined as follows: each possible fragment (segment pair) f is assigned a *weight score* $w(f)$ depending on the probability $P(f)$ of *random occurrence* of such a fragment. More precisely, the program uses a similarity function s assigning a score $s(a, b)$ to each possible pair (a, b) of residues. For protein alignment, one of the usual substitution matrices can be used; for alignment of DNA or RNA sequences, the program simply distinguishes between matches and mismatches. For a fragment f, its *Needleman-Wunsch score* $NW[f]$ is calculated that is defined as the sum of similarity values of aligned nucleotides or amino acid residues (note again that fragments do not contain gaps). To define the weight score $w(f)$ of f, we consider the probability $P(f)$ of finding a fragment f' of the same length as f and with a Needleman-Wunsch score $NW[f'] \geq NW[f]$ in *random sequences* of the same length as the input sequences. $w(f)$ is then defined as the *negative logarithm* of this probability; see [51] for more details. The *total* score of a – pairwise or multiple – alignment is defined as the *sum of weight scores* of the fragments it is composed of; gaps are not penalised. The idea is that the *less* likely a given fragment collection occurs just by chance, the *more* likely it is biologically relevant so the higher its score should be. Thus, while standard alignment approaches try to find an alignment that is *most likely* under the assumption that the input sequences are related by common ancestry [23], we try to find an alignment that is *most unlikely* under the assumption that the sequences are *not* related. A pairwise alignment in the sense of the above definition corresponds to a *chain* of fragments, and an alignment with maximum total weight score can be found using a recursive fragment-chaining procedure [55]; for multiple alignment, a *greedy* heuristic is used [1, 51].

As explained above, DIALIGN defines the score $S(A)$ of an alignment $A = \{f_1, \ldots, f_k\}$ as the sum of weight scores $w(f_i)$ of its constituent fragments, and these weight scores are, in turn, defined as negative logarithms of probabilities

$P(f_i)$ of their random occurrence. Thus, the score $S(A)$ is calculated as

$$S(A) = \sum_{f \in A} w(f) = \sum_{f \in A} -\log P(f) = -log \prod_{f \in A} P(f)$$

and searching for an alignment with *maximal* score is equivalent to searching for a consistent collection of fragments $A = \{f_1, \ldots, f_k\}$ with a *minimal* product of probabilities $\prod_{f \in A} P(f)$. But, considering the *product* of fragment probabilities means considering the probability of their *joint* occurrence under the assumption that these events are *independent* of each other. This would be reasonable if we searched for an *arbitrary* fragment collection with low probability of random occurrence. In our approach, however, we require a fragment collection to be *consistent*, so the set of allowed combinations of fragments is drastically reduced. The probability of finding a *consistent* set of fragments is consequently far smaller than the product of the probabilities of finding all of the corresponding *individual* fragments. Thus, by using the product $\prod_{f \in A} P(f)$, DIALIGN generally *over-estimates* the probability $P(A)$ of random occurrence of an alignment A.

In our context, the crucial point is that the probabilities $P(A)$ – and therefore the scores $S(A)$ – are not *uniformly* over-estimated – or under-estimated, respectively – for all possible alignments, but that there is a large difference between global and local alignments. For a *global* alignment A_g that covers most of the sequences, the *discrepancy* between the *real* probability $P(A_g)$ of its random occurrence and the approximation $\prod_{f \in A_g} P(f)$ used by DIALIGN is far more significant than for a *local* alignment A_l. This is because a global alignment corresponds to a *dense* collection of fragment, so here the consistency constraints are much tighter than in a local alignment consisting of only a few isolated fragments. As a result, DIALIGN *relatively* over-estimates the probability $P(A_g)$ of a global aligment A_g compared with an alternative local alignment A_l, so it *under-estimates* the score $S(A_g)$ compared with the score $S(A_l)$.

3.1.2 Approximation of the objective function

It is very time-consuming to compute the weight score for each fragment exactly, therefore, DIALIGN 2.2 and DIALIGN-T use approximative calclulations with sufficient accuracy and with a reasonable running time.

The previous implementation, DIALIGN 2.2, uses pre-calculated probability tables to calculate fragment weight scores; these tables are based on the BLOSUM 62 substitution matrix. They were calculated years ago and are difficult to re-calculate if a user wants to employ another similarity matrix. It is therefore not possible to run DIALIGN 2.2 with substitution matrices other than BLOSUM 62. In DIALIGN-T, we use a rather efficient way estimating the probabilities that are used for our weight score calculations. We pre-calculated probability tables for a variety of substitution matrices. In addition, the user can re-calculate these tables 'on the fly' for arbitrary matrices with a moderate increase in program running time.

As explained in section 3.1.1, we define the weight score of a fragment f involving sequences S_i and S_j as

$$w(f) = -logP(f)$$

where $P(f)$ denotes the probability of the occurrence of a fragment f' of the same length as f and with Needleman-Wunsch score $NW[f'] \geq NW[f]$ in *random sequences* of the same length as S_i and S_j. By *random sequences* we mean *independent identically distributed (iid)* sequences where each residue occurs at any position with probability 1/4 for nucleic acid sequences and 1/20 for protein sequences. In the following, we outline how our program approximates the probabilities $P(f)$.

In the first step, we estimate the probability $\tilde{P}(s,n)$ of finding a fragment f' of length n and with Needleman-Wunsch score $NW[f'] \geq s$ in random sequences of length $2 \cdot n$. Note that $\tilde{P}(s,n)$ depends on the underlying substitution matrix but not on the length or composition of the input sequences S_i and S_j. The numerical values $\tilde{P}(s,n)$ are estimated as follows:

1. Random experiments are performed to obtain preliminary estimates $\tilde{P}_{exp}$ for $\tilde{P}$. The experimental values $\tilde{P}_{exp}$ should approximate $\tilde{P}$ with sufficient accuracy for values of n and s where enough experimental data is available. This is the case if $\tilde{P}(s,n)$ is not too small.

2. For small values of $\tilde{P}(s,n)$, we first compute the probability $P_1(s,n)$ for a *single* random fragment f' of length n to have a Needleman-Wunsch score $NW[f'] \geq s$. $P_1(s,n)$ can be easily calculated as a sum of convolution products using an iterative procedure that computes $P_1(s,n)$ from all pairs of $(P_1(s_1,n), P_2(s_2,n))$ with $s_1 + s_2 = s$. Similar to [51], small values of $\tilde{P}$ are estimated using the approximation formula

$$\tilde{P}(s,n) \approx P_1(s,n) \cdot (n+1)^2.$$

3. All in all, we define $\tilde{P}$ for a given value s by first considering the trivial case $n = 1$ and then defining for $n = 2, \ldots, M$:

$$\tilde{P}(s,n) = \begin{cases} P_1(s,n) \cdot (n+1)^2 & \text{if } P_1(s,n) \cdot (n+1)^2 < \tilde{P}(s,n-1) \\ P_{exp}(s,n) & \text{else.} \end{cases}$$

The procedure described for estimating $\tilde{P}(s,n)$ is computationally demanding. Since the values $\tilde{P}(s,n)$ do not depend on the input sequences, we *pre-calculated* these probabilities for several standard substitution matrices and stored their values in auxiliary files from which they are retrieved during the program run.

In the second step, we use $\tilde{P}(s,n)$ to estimate the probability $P(s,n)$ of finding a fragment f' of length n with Needleman-Wunsch score $NW[f'] \geq s$ in sequences of the same length as the input sequences. This step is computationally less expensive and can therefore be carried out during the program run. Let l_i and l_j be the lengths of the input sequences S_i and S_j, respectively. Similar to [51], we compute $P(s,n)$ as

$$P(s,n) = \begin{cases} 1 - (1 - \tilde{P}(s,n))^{l_i l_j / (4n^2)} & \text{if this value is} > P_T \\ \tilde{P}(s,n) \cdot l_i \cdot l_j / (4n^2) & \text{else.} \end{cases}$$

where P_T is a threshold parameter. During a program run, the values $P(s,n)$ are calculated for all possible values of n and s *before* the pairwise alignment of

sequences S_i and S_j is carried out. The negative logarithms

$$-\log P(s,n)$$

are stored in a look-up table from which they are retrieved during the pairwise alignment to define the fragment scores.

We pre-calculated the probabilities $\tilde{P}(s,n)$ for several substitution matrices of the BLOSUM family. To determine the experimental probability values $P_{exp}(s,n)$, we carried out 10^6 random experiments for each relevant pair of parameters (s,n). Here, we considered values for n between 1 and a maximum fragment length $M = 100$. Files with the resulting values of $\tilde{P}(s,n)$ values are delivered together with the DIALIGN-T software package. To calculate $P(f)$, we use a threshold probability $P_T = 10^{-8}$. Our program can also be used to calculate the values $\tilde{P}(s,n)$ for *arbitrary* user-defined substitution matrices. Calculating these values using 10^5 random experiments for each value of n and s takes around 20 minutes on a Linux workstation (RedHat 8.0) with an 1.5 Ghz Pentium 4 processor and 512 MB Ram. In our experience, 10^5 random experiments are sufficient to obtain high-quality probability estimates.

3.1.3 Dynamic programming

Altogether the initial stage of DIALIGN-T consists of computing for each pair s_i, s_j of the input sequences $s_1, \ldots, s_k$ an optimum pairwise alignment as a set of fragments $F_{i,j}$ that optimize the sum of weight scores with respect to the objective function described in subsection 3.1.1

$$\sum_{f \in F_{i,j}} w(f) \rightarrow \max$$

In DIALIGN-T we use the space-efficient pairwise dynamic programming algorithm also implemented in DIALIGN 2.2 and proposed in [54, 55]. This approach essentially consumes only linear space in the general case and is therefore very suitable for large input sequences.

Let $t = t_1, \ldots, t_m$ and $u = u_1, \ldots, u_n$ be two input sequences for which we construct a comparison matrix that is very much similar to the similarity matrix (see chapter 2) and analogously is also indexed by all possible prefix pairs of t and u. This matrix is processed in a column-by-column manner from left to right. Let F denote the set of all possible fragments between t and u and for each fragment $f \in F$ that starts in the currently processed column i we store the weight score of an optimum alignment ending in f together with a pointer to the predecessor of f (i.e. the second-last fragment in the alignment) in a linked list $F_{i'}$ associated with column i' where f ends. Assuming that we know all optimum alignments for all prefixes $t_1 \ldots t_{i-1}$ and $u_1 \ldots u_j, 1 \leq j \leq n$ in column $i-1$, we can easily compute this optimum alignment for f.

Hence we proceed as follows from left to right - column by column. For the current column i we compute all optimum alignments for fragments that start in column i by extending the optimum alignments from the previous column $i-1$. Then we compute the optimum alignments for column i by again using column $i-1$ and the linked list F_i, as follows: Either the optimum alignment

for prefix $t_1 \dots t_i$ and $u_1 \dots u_j$ is determined by the optimum alignment for the prefix $t_1 \dots t_{i-1}$ and $u_1 \dots u_j$ stored in the previous column or it is given by the highest scoring alignment in F_i ending in t_i and u_l with an $l \leq j$. After having computed the alignments for column i we forget about column $i-1$ and the list F_i since they won't be needed in further steps. After all columns have been processed, we apply a trace-back beginning at the last entry of column m in order to retrieve the optimum pairwise alignment.

Lemma 3.1
The overall runtime for computing all pairwise alignments is then $O(k^2 n^2 \cdot L_{max})$ for k input sequences having maximum sequence length n and a maximum fragment length of L_{max}.

In DIALIGN-T the maximum fragment length is bounded by 100 so the overall number of fragments looked at is $m \cdot n \cdot 100$ when computing a pairwise alignment of two sequences of length m and n. In practice the algorithm does not consider fragments having a computed weight score $w(f) = 0$, which applies to most of the fragments yielding a significant speed-up. So far the dynamic programming stage is similar to DIALIGN 2.2 [54, 55] and in the next subsection we will explain the qualitative improvements within DIALIGN-T for this stage.

3.1.4 Excluding low-scoring sub-fragments

In the previous subsections, we systematically explained the objective function and the dynamic programming algorithm used in DIALIGN 2.2. We observe that DIALIGN 2.2 is biased towards isolated local similarities due to the objective function used. If the program can choose between (a) a *global* pairwise alignment consisting of many fragments with low individual fragment scores and (b) an alternative *local* alignment consisting of only a few isolated fragments with higher individual scores, it tends to prefer the second type of alignment. Consequently, for sequences with weak but global similarity, DIALIGN is vulnerable to spurious random similarities.

An improved objective function that used a better approximation to the probability $P(A)$ of random occurrence of an alignment A would have to take into account the combinatorial constraints given by our consistency condition. Defining such an objective function would be mathematically challenging and would drastically deteroriate the running time of the program. For DIALIGN-T, we therefore use the objective function that has been used in previous versions of DIALIGN. However, we introduce two heuristics to counterbalance the bias in this objective function towards isolated local alignments: One is the exclusion of low-scoring sub-fragments described in this section and the other is the introduction of weight score factors, which will be described in subsection 3.2.2.

The pairwise alignment algorithm that we are using is a modification of the space-efficient fragment-chaining algorithm described in the previous subsection. At each position (i, j) in the comparison matrix, this algorithm considers all fragments (= segment pairs) starting at (i, j) up to a certain maximum length M. For protein alignment, the previous program DIALIGN 2.2 uses a default value of $M = 40$; M can be reduced to speed up the program, but this may result

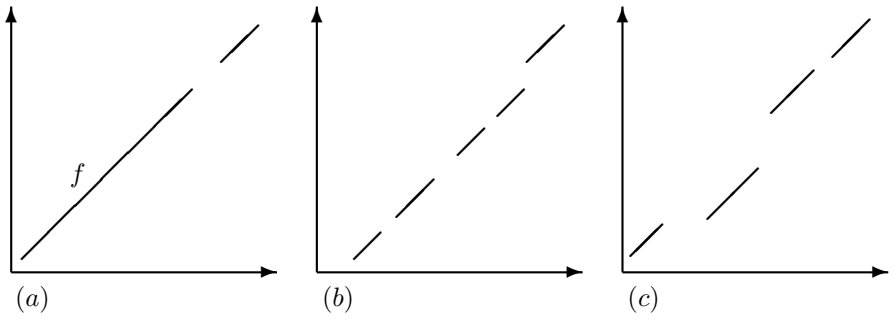

Fig. 3.1: Exclusion of low-scoring regions from alignment fragments. The scoring scheme used in DIALIGN gives relatively high weight scores to single fragments with high Needleman-Wunsch scores (a). In our new approach, we exclude low-scoring sub-regions within long fragments by applying a stop criterion for fragment extension. This can result in the replacement of a long fragment f by multiple sub-fragments (b) or in a completely different alignment (c).

in decreased alignment quality. Initially, the length limitation for fragments has been introduced to reduce the program running time; in this way the time complexity of the pairwise fragment-chaining algorithm is reduced from $O(l^3)$ to $O(l^2)$ where l is the maximum length of the two sequences. One might think that increasing the maximum fragment length M would result in improved alignment quality, but in fact, we observed that with slightly increased values for M, better alignments were obtained. However, with values $M > 50$, the quality of the produced alignments decreased dramatically.

In systematic test runs, we observed that for large values of M, output alignments often contain long fragments involving a mixture of high-scoring and low-scoring sub-fragments. With an ideal objective function, a single long fragment f containing low-scoring sub-fragments would automatically receive a *lower* score than the chain of short fragments that would be obtained from f by removing those low-scoring sub-fragments. As a result, output alignments would tend to consist of shorter fragments rather than of longer fragments with low-scoring sub-regions. For reasons explained in the previous section, however, the scoring scheme used by DIALIGN over-estimates single long fragments compared with chains of smaller fragments that would be obtained by removing low-scoring regions from those long fragments.

In our new approach, we use the following heuristics to prevent the algorithm from selecting long fragments with low-scoring sub-regions. We define a length threshold L for low-quality sub-fragments. Sub-fragments of length $\geq L$ with negative Needleman-Wunsch scores are allowed within short fragments but are excluded in fragments of length $\geq T$ where $T < M$ is a parameter that can be adjusted by the user. For a pair of input sequences S_1 and S_2 and given values for the parameters T, M and L, our new algorithm proceeds as follows. Let $f(i, j, k)$ denote the fragment of length k that starts at position i in sequence S_1 and at position j in sequence S_2, respectively. By $S_1[k]$, we denote the k-th character in sequence S_i. As in the original DIALIGN algorithm, we traverse the comparison matrix for S_1 and S_2, and at every position (i, j), we consider fragments *starting* at this position; suitable fragments are then added to a growing

set F of candidate fragments from which the algorithm selects a fragment chain with maximum total score with respect to the underlying objective function [55]. If a region of low quality occurs, the maximum fragment length $M(i,j)$ for fragments starting at (i,j) is reduced from M to T. More formally, we perform the following steps for fragments starting at a fixed position (i,j):

1. Initially, the maximum length for fragments starting at (i,j) is $M(i,j) = M$.
2. We start with length $k = 1$, i.e. we consider the fragment $f(i,j,1)$.
3. If the current fragment length k exceeds $M(i,j)$ then the procedure stops and we continue with fragments starting at $(i,j+1)$.
4. If the similarity score $s(S_1[i+k-1], S_2[j+k-1])$ of the last residue pair in $f(i,j,k)$ is not negative, we take the fragment $f(i,j,k)$ into account by adding it to the set F and continue with step 7. Otherwise, we detect the potential beginning of a low-quality sub-fragment starting at positions $i+k-1$ and $j+k-1$.
5. In this case we do a look-ahead and calculate the NW-score of the potential low-scoring fragment $f(i+k-1, j+k-1, L)$, which is defined as

$$NW[f(i+k-1, j+k-1, L)] = \sum_{p=0}^{L-1} s(S_1[i+k-1+p], S_2[j+k-1+p]).$$

6. If $NW[f(i+k-1, j+k-1, L)] < 0$, we actually detect a low-quality sub-fragment. If $k > T$, the procedure stops and no further increase of k is considered, otherwise we set $M(i,j) = T$.
7. The length k is incremented by 1 and we continue with the step 3.

By default, our program uses a length threshold for low-quality sub-fragments of $L = 4$ and the maximum length of fragments containing such regions of low quality is $T = 40$. These values have been determined based on systematic test runs on BAliBASE. At this point, we want to mention the impact of the parameters L and T on the quality of the produced output alignments. For example, with values $L = 3$ or $L = 5$, the alignment quality is dramatically worsened compared with the default value $L = 4$.

Our stop criterion for low-scoring sub-fragments not only improves the quality of the resulting alignments but also reduces the program running time. The runtime of our pairwise algorithm is proportional to the number of fragments that are considered for alignment. Thus, the *worst-case* time complexity is $O(l_1 \cdot l_2 \cdot M)$ where l_1 and l_2 are the lengths of the input sequences. By excluding long fragments with low-scoring sub-fragments, we ignore a large number of fragments that would have been considered for alignment in previous program versions. Therefore, our new heuristics allow us to *increase* the maximum possible fragment length from $M = 40$ to $M = 100$ without excessively increasing the total number of fragments that are to be looked at. A further extension of

M is prohibited due to numerical instabilities. Altogether, the resulting alignments can reflect the extension of existing homologies more realistically than the previous version of DIALIGN with only a moderate increase in program running time.

3.2 Assembling the multiple sequence alignment

Having computed the set of all fragments from the pairwise alignment phase, we will now describe how we assemble them in DIALIGN-T. As in the previous version, DIALIGN 2.2, the idea is to build the alignment greedily, however, with some new improvements.

3.2.1 Consistency data structure

DIALIGN-T starts with an empty alignment A and successively adds consistent fragments to it. By aligning two positions of two sequences we naturally impose constraints for other pairs of positions to be alignable in order to receive an overall consistent multiple sequence alignment. We keep track of those constraints in the data structure of predecessor and successor frontiers that are introduced in [1] and also used being used in DIALIGN 2.2. Let $s_1, \ldots, s_k$ be the set of input sequences and A the initial alignment of those sequences without any gaps. We define the two functions $PredF_A$ and $SuccF_A$ that assign for each position r in sequence i the minimum and maximum position in sequence j that it can be aligned consistently to in A, i.e. $PredF_A(i, r, j)$ is the minimum position in sequence j that the position r of i can be consistently aligned to without destroying any alignment of positions in A; $SuccF_A(i, r, j)$ is defined analogously, but instead of the minimum possible position in sequence j it is set to the maximum position in sequence j. We initialize $PredF_A$ with -1 and $SuccF_A(\cdot, \cdot, j)$ with $|s_j|+1$. Obviously we have $PredF_A(i, r, j) = SuccF_A(i, r, j) = p$ if position r of sequence i is aligned with position p of sequence j. If position r of sequence i is not aligned to any position of sequence j, the value of $PredF_A(i, r, j)$ is the maximum position in sequence j that is aligned in A to any previous position $< r$ in sequence i plus 1; analogously $Succ_F(i, r, j)$ is the minumum position in sequence j that is aligned to any subsequent position $> r$ in sequence i minus 1.

We assume that we have calculated all $PredF_A$ and $SuccF_A$ with respect to alignment A and found that aligning position r of sequence i with position p in sequence j is consistent to A, i.e.

$$\begin{array}{ll} PredF_A(i, r, j) & \leq p \\ SuccF_A(i, r, j) & \geq p. \\ PredF_A(j, p, i) & \leq r \\ SuccF_A(j, p, i) & \geq r, \end{array}$$

whereby the first two inequalities are true if and only if the last two are true and vice versa due to the definition of $PredF_A$ and $SuccF_A$. We also conclude that checking if position r in sequence i is alignable to position p in sequence j can be done in time $O(1)$ if the predecessor and successor frontier data structure

is in place. If we now align those two positions in A and call the resulting alignment A' we have to update $PredF_A$ to $PredF_{A'}$ and $SuccF_A$ to $SuccF_{A'}$ appropriately. We do this by iterating through all possible positions r' and all sequence pairs $1 \leq i', j' \leq k$ with $i' \neq j', 1 \leq r' \leq |s_{i'}|$ and setting:

$$PredF_{A'}(i', r', j') = \begin{cases} \max\{PredF_A(i', r', j'), PredF_A(j, p, j')\} & \text{if } PredF_A(i', r', i) >= r \\ \max\{PredF_A(i', r', j'), PredF_A(i, r, j')\} & \text{if } PredF_A(i', r', j) >= p \\ PredF_A(i', r', j') & \text{else} \end{cases}$$

and

$$SuccF_{A'}(i', r', j') = \begin{cases} \min\{SuccF_A(i', r', j'), SuccF_A(j, p, j')\} & \text{if } SuccF_A(i', r', i) <= r \\ \min\{SuccF_A(i', r', j'), SuccF_A(i, r, j')\} & \text{if } SuccF_A(i', r', j) <= p \\ PredF_A(i', r', j') & \text{else.} \end{cases}$$

Of course we only have to update $PredF_A$ and $SuccF_A$ in all but the above mentioned 'else'-case. To further speed up the update of the frontier data structure, we additionally store a bit-valued orphan flag for each position r in every sequence i indicating whether this position has been aligned at all to any other position in any other sequence or not. For any orphan position in sequence i, we do not maintain the data structure but we store a pointer to the equivalent $PredF_A$ of the immediately previous position in sequence i that is aligned to any other position and a pointer to the equivalent $SuccF_A$ to the next immediate position that has already been aligned.

For the computational complexity of the update of this data structure we conclude:

Theorem 3.1
Let $s_1, \ldots, s_k$ be k sequences over an alphabet Σ and $\hat{A}$ the empty (trivial) alignment over those sequences. The initial frontiers $PredF_{\hat{A}}$ and $SuccF_{\hat{A}}$ can be computed in time $O(n \cdot k^2)$, whereby n denotes the length of the longest input sequence.

Let A be an arbitrary and consistent alignment on $s_1, \ldots, s_k$ and $PredF_A$ and $Succ_F$ the respective frontiers. Furthermore, let (i, p) and (j, r) be a position p in sequence i and a position r in sequence j, respectively. Checking whether (i, p) and (j, r) can be consistently aligned in A can be solved in time $O(1)$ and in the positive case the respective update of $PredF_A$ to $PredF_{A'}$ and $SuccF_A$ to $SuccF_{A'}$ can be done in time $O(n \cdot k^2)$ whereby A' is the alignment A plus the alignment of (i, p) with (j, r).

If we now want to add a fragment f of length l starting at position p in sequence i and at position r in sequence j to an alignment A, we iterate through all pairs of positions $(i + t, j + t)$ with $0 \leq t < l$ and check whether they are alignable according to $PredF_A$ and $SuccF_A$. If so, we can add fragment f to the alignment A by again iterating through the same l pairs of positions and updating the predecessor and successor frontier functions accordingly for each pair.

3.2.2 Weight score factors

As mentioned above, DIALIGN uses a greedy optimisation procedure for multiple alignment. The order in which fragments are included into the multiple

alignment is determined based on their *weight scores*. A general problem with this greedy approach is that if a erroneous fragment is accepted for multiple alignment, it cannot be removed later on. Note that even a single wrong choice in the greedy procedure can impair the quality of the resulting alignment dramatically. Thus, special care has to be taken to prioritise fragments for the greedy algorithm. We observed that in many cases spurious but high-scoring fragments from pairwise alignments are *inconsistent* with a good overall multiple alignment. Due to their weight scores, however, such fragments may be incorporated into the multiple alignment by the original DIALIGN, thereby leading to output alignments of lower quality.

As explained in subsection 3.1.4, the weight score of a fragment depends on the probability of its *random occurrence* in sequences of the same length as the input sequences. Thus, weight scores are based purely on *intrinsic* properties of fragments – and on the length of the input sequences – but they do not take into account the *context* of a fragment within the pairwise alignment. In reality, however, the context of a fragment is crucial to assess its reliability. If a fragment f is part of a high-scoring pairwise alignment, then f is, of course, far more likely to be biologically significant than if the same fragment f were found isolated in otherwise un-related sequences. Therefore, the overall similarity between two sequences should be taken into account if fragments are ranked prior to the greedy procedure.

In DIALIGN-T, we adopt the following approach: we multiply the weight score of each fragment by the square of the total weight score of the respective sequence pair divided by the overall weight score of all pairwise alignments. Let $S_1, \ldots, S_n$ be the input sequences and let f be a fragment involving sequences S_i and S_j. Next, let $w(S_i, S_j)$ denote the total weight score of the pairwise alignment for S_i and S_j – i.e. the sum of weight scores of an optimal chain of fragments – and let W be the total sum of weight scores of all pairwise alignments. That is, we define

$$W = \sum_{1 \leq i < j \leq n} w(S_i, S_j).$$

We then define the *adjusted* weight score

$$w'(f) = w(f) \cdot \left(\frac{w(S_i, S_j)}{W} \right)^2$$

and in our greedy algorithm, fragments are sorted according to their adjusted scores $w'(f)$. In this way, we prefer fragments belonging to sequence pairs of high similarity to those from weakly related sequence pairs. Altogether, this weight adjustment respects the similarity of the sequence pairs better than the previous method and hence may keep the greedy procedure from adding isolated spurious fragments that would have led to a lower-scoring and biologically less meaningful output alignment. The sorted list of fragments from the optimal pairwise alignments is kept in a binary heap structure that can be updated efficiently when inconsistent fragments are removed or modified as explained in the next subsection.

3.2.3 Dealing with inconsistent fragments

In the original DIALIGN approach, an inconsistent fragment f is *completely* discarded in the greedy procedure, even if just a few residue pairs are inconsistent with the current alignment. In such a situation, it would of course be more sensible to remove only those inconsistent residue pairs from f and give the remaining sub-fragments a second chance in the greedy selection process. It is easy to see that a fragment f is consistent with an existing alignment A if and only if each *pair* of aligned residues in f is consistent with A. In our new implementation, we use the following procedure for non-consistent fragments. An inconsistent fragment f is processed from left to right. Starting with the left-most residue pair, we remove all inconsistent residue pairs until we find the first consistent pair p. Next, we consider all consistent residue pairs starting with p until we again find an inconsistent residue pair. In this way, we obtain a consistent sub-fragment f' of f for which we calculate the weight score $w(f')$. By construction, f' is consistent with the existing alignment and could, in principle, be added to the list of accepted fragments.

However, we do not immediately include f' into the growing multiple alignment since the score $w(f')$ might be smaller than the original score $w(f)$. Instead, we insert f' at the appropriate position in our sorted list of fragments depending on its adjusted weight score $w'(f')$. We therefore use a binary heap structure such that consistent sub-fragments of inconsistent fragments can be efficiently repositioned according to their newly calculated adjusted weights. The remainder of f is treated accordingly, i.e. inconsistent residue pairs are removed and the remaining consistent sub-fragments are inserted at appropriate positions in the list of candidate fragments. Note that with our weighting function w, the weight score $w(f')$ of a sub-fragment f' contained in a fragment f can, in general, be *larger* than the weight $w(f)$. In the above situation, however, we necessarily have $w(f') \leq w(f)$ [and therefore $w'(f') \leq w'(f)$] since we assumed that f is part of the optimal pairwise alignment of two sequences. If the score $w(f')$ of a sub-fragment of f exceeded $w(f)$, then f' would have been selected for the optimal pairwise alignment instead of f.

3.3 Benchmark results

We evaluated the performance of our program and compared it to alternative multi-alignment software tools using a wide variety of benchmark sequences. As a first set of reference data, we used the well-known BAliBASE 2.1 [78]. BAliBASE has been used in numerous studies to test the accuracy of multiple-protein-alignment software. It should be mentioned that, although some of the reference sequences in BAliBASE contain insertions and deletions of moderate size, BAliBASE is heavily biased towards *globally* related protein families. All BAliBASE sequences contain homologous *core blocks* with verified 3D structure; alignment programs are evaluated according to their ability to correctly align these blocks. According to the BAliBASE authors, these core blocks cover 58 % of the residues in the database. However, sequence similarity is clearly not restricted to those regions of verified 3D structure so, in reality, far more than 58 % of the total sequence length are homologous to other sequences in the respective

sequence families. Also, the sequences in BAliBASE are not realistic full-length sequences, but they have been truncated by the BAliBASE developers in order to remove non-related parts of the sequences. As a result, BAliBASE consists almost entirely of *globally* related sequence sets; this is why *global* alignment programs such as CLUSTAL W perform best on these benchmark data.

To study the performance of alignment programs on *locally* related sequence sets, Lassmann and Sonnhammer used artificial random sequences with implanted conserved motifs [46]. Random sequences are frequently used to evaluate computational sequence analysis tools; they are particularly useful to study the *specificity* of a tool, see e.g. [73, 36, 63]. Unfortunately, the benchmark data by Lassmann and Sonnhammer are not publicly available. Therefore, we set up our own benchmark database for local multiple protein alignment that we called *IRMBASE* (Implanted Rose Motifs Base).

As Lassmann and Sonnhammer did in their previous study, we produced groups of artificial conserved sequence motifs using the ROSE software tool [73]. ROSE simulates the process of molecular evolution. A set of 'phylogenetically' related sequences is created from a user-defined 'ancestor' sequence according to a phylogenetic tree. During this process, sequence characters are randomly inserted, deleted and substituted under a pre-defined stochastic model. In this way, a sequence family with known 'evolution' is obtained, so the 'correct' multiple alignment of these sequences is known. Note that these alignments contain mismatches as well as gaps. We inserted families of conserved motifs created by ROSE at randomly chosen positions into non-related *random* sequences. In this way, we produced three reference sets, *ref1, ref2* and *ref3*, of artificial protein sequences. Sequences from ref1, ref2 and ref3 contain one, two and three motifs, respectively. Each reference set consists of 60 sequence families, 30 of which contain ROSE motifs of length 30 while the remaining 30 families contain motifs of length 60. Twenty sequence families in each of the reference sets consist of 4 sequences each, another 20 families consist of 8 sequences while the remaining 20 families consist of 16 sequences. In ref1, random sequences of length 400 are added to the conserved ROSE motif while for ref2 and ref3, random sequences of length 500 are added.

For both BAliBASE and IRMBASE, we used two different criteria to evaluate multi-alignment software tools. We used the *sum-of-pair score*, where the percentage of correctly aligned *pairs* of residues is taken as a quality measure for alignments. In addition, we used the *column score* where the percentage of correct *columns* in an alignment is the criterion for alignment quality. Both scoring schemes were restricted to *core blocks* within the reference sequences where the 'true' alignment is known. For IRMBASE, the core blocks are defined as the conserved ROSE motifs. In general, the sum-of-pairs score is more appropriate than the column score because this latter score ignores all correctly aligned residues in an alignment column if a single residue in this column is mis-aligned. However, there are situations where the column score is more meaningful than the sum-of-pairs score. This is the case, for example, for BAliBASE reference sets containing 'orphan sequences'.

To compare the output of different programs to the respective benchmark alignments, we used C. Notredame's program `aln_compare` [60]. Tables 3.1 and 3.2 summarise the performance of DIALIGN-T, DIALIGN 2.2, CLUSTAL W,

Method	ref1	ref2	ref3	Total
DIALIGN-T	94.07%	92.69%	92.68%	93.14%
DIALIGN 2.2	92.26%	92.72%	91.87%	92.28%
T-COFFEE 1.37	91.18%	85.61%	87.81%	88.20%
PROBCONS 1.09	66.74%	68.30%	77.92%	70.98%
POA V2	90.26%	43.61%	36.85%	56.91%
MUSCLE 3.5	36.16%	37.84%	52.30%	42.10%
CLUSTAL W 1.83	8.02%	12.69%	20.16%	13.62%

Tab. 3.1: Performance of seven protein multi-alignment programs on the IRMBASE 1.0 database of benchmark alignments. Percentage values are *sum-of-pairs scores*, i.e. the percentage of correctly aligned *residue pairs* of ROSE motifs contained in the IRMBASE sequence families.

MUSCLE, PROBCONS, T-COFFEE and POA on IRMBASE while Tables 3.3 and 3.4 show their accuracy on BAliBASE. In addition, Tables 3.5, 3.6, 3.7 and 3.8 contain the percentage of sequence sets where DIALIGN-T is outperformed by the other programs that we tested. Tables 3.1 and 3.2 show that, on locally related sequence families, DIALIGN-T is significantly superior to the algorithms DIALIGN 2.2, T-COFFEE, MUSCLE, POA and CLUSTAL W. Only DIALIGN-T, DIALIGN 2.2, T-COFFEE and (in a very reduced way) PROBCONS produced reasonable results on IRMBASE 1.0. However, DIALIGN-T is the fastest and most accurate amongst all methods that we looked at. We would like to emphasize that the performance of multi-alignment methods on *simulated* data only roughly reflects their performance on *real* data. Nevertheless, in the absence of real-world benchmark data for local multiple alignment, the results on IRMBASE can give us an idea of how different algorithms deal with locally conserved motifs.

For globally related sequence families, Tables 3.3 and 3.4 show that, on average, DIALIGN-T outperforms DIALIGN 2.2 and POA on BAliBASE 2.1 while its performance is similar to CLUSTAL W. By contrast, the previous version DIALIGN 2.2 is clearly outperformed by CLUSTAL W on these data sets. Finally, DIALIGN-T is still outperformed on many of the BAliBASE test sequences by T-COFFEE, MUSCLE and PROBCONS; the latter-most program is currently the best-performing multiple aligner on BAliBASE. The superiority of our new approach compared to DIALIGN 2.2 and POA is clearly *statistically significant* according to the Wilcoxon Matched Pairs Signed Rank Test [87]. On BAliBASE reference sets ref1, ref2 and ref3, where sequences contain only small insertions and deletions, the performance of DIALIGN-T is roughly comparable to CLUSTAL W, but still significantly worse than T-COFFEE, PROBCONS or MUSCLE. Our program is statistically significantly superior or equal to all

Method	ref1	ref2	ref3	Total
DIALIGN-T	82.28%	78.36%	79.71%	80.12%
DIALIGN 2.2	79.46%	77.82%	78.24%	78.51%
T-COFFEE 1.37	75.35%	66.60%	69.21%	70.19%
PROBCONS 1.09	33.13%	37.95%	51.26%	40.78%
POA V2	73.00%	12.46%	07.45%	30.97%
MUSCLE 3.5	09.41%	10.89%	22.37%	14.22%
CLUSTAL W 1.83	00.00%	00.83%	05.14%	01.92%

Tab. 3.2: Performance of seven protein multi-alignment programs on IRMBASE using *column scores* as the quality criterion. Thus, percentage values denote the percentage of correct *alignment columns* of the ROSE motifs in IRMBASE.

Method	ref1	ref2	ref3	ref4	ref5	Total
DIALIGN-T	82.76%	91.28%	75.34%	86.43%	93.30%	84.69%
DIALIGN 2.2	81.40%	89.56%	68.93%	91.24%	94.14%	83.59%
T-COFFEE 1.37	84.67%	93.24%	80.32%	75.80%	96.20%	85.95%
PROBCONS 1.09	90.37%	94.61%	84.34%	89.20%	98.07%	91.11%
POA V2	74.66%	88.32%	63.14%	82.62%	76.71%	76.76%
MUSCLE 3.5	88.25%	93.59%	82.36%	85.62%	97.80%	89.21%
CLUSTAL W 1.83	86.43%	93.22%	75.79%	81.09%	86.10%	86.15%

Tab. 3.3: Performance of seven protein multi-alignment programs on the BAliBASE benchmark database using *sum-of-pairs scores* as the evaluation criterion.

Method	ref1	ref2	ref3	ref4	ref5	Total
DIALIGN-T	73.22%	43.43%	44.69%	66.13%	77.05%	65.65%
DIALIGN 2.2	71.49%	37.42%	35.03%	81.88%	84.47%	64.82%
T-COFFEE 1.37	75.32%	53.44%	52.20%	45.09%	86.96%	68.20%
PROBCONS 1.09	83.21%	59.76%	61.34%	71.09%	91.86%	77.23%
POA V2	63.21%	39.02%	25.57%	57.22%	47.18%	54.18%
MUSCLE 3.5	80.79%	56.37%	56.74%	62.65%	91.57%	74.13%
CLUSTAL W 1.83	78.39%	56.24%	48.87%	50.44%	63.89%	68.48%

Tab. 3.4: Performance of seven protein multi-alignment programs on BAliBASE using *column scores*.

Method	ref1	ref2	ref3	Total
DIALIGN 2.2	$20.00\%^{+}$	$23.33\%^{0}$	$23.33\%^{+}$	$22.22\%^{+}$
T-COFFEE 1.37	$40.00\%^{0}$	$31.67\%^{+}$	$41.67\%^{+}$	$37.78\%^{+}$
PROBCONS 1.09	$20.00\%^{+}$	$15.00\%^{+}$	$21.67\%^{+}$	$18.89\%^{+}$
POA V2	$16.67\%^{+}$	$0.00\%^{+}$	$0.00\%^{+}$	$5.55\%^{+}$
MUSCLE 3.5	$5.00\%^{+}$	$5.00\%^{+}$	$0.00\%^{+}$	$3.33\%^{+}$
CLUSTAL W 1.83	$0.00\%^{+}$	$0.00\%^{0}$	$0.00\%^{0}$	$0.0\%^{+}$

Tab. 3.5: Percentage of sequence families where DIALIGN-T is outperformed on IRMBASE 1.0 by alternative methods according to the *sum-of-pairs score*. The symbol $^{+}$ denotes statistically significant superiority, $^{-}$ statistically significant inferiority and ^{0} non-significant superiority or inferiority of DIALIGN-T. Significance has been calculated according to the Wilcoxon Matched Pairs Signed Rank Test with $p \leq 0.05$.

Method	ref1	ref2	ref3	Total
DIALIGN 2.2	11.67%$^{+}$	21.67%^{0}	23.33%$^{+}$	18.89%$^{+}$
T-COFFEE 1.37	36.67%^{0}	30.00%$^{+}$	26.67%$^{+}$	31.11%$^{+}$
PROBCONS 1.09	18.33%$^{+}$	01.67%$^{+}$	16.67%$^{+}$	16.67%$^{+}$
POA V2	15.00%$^{+}$	00.00%$^{+}$	00.00%$^{+}$	05.00%$^{+}$
MUSCLE 3.5	05.00%$^{+}$	05.00%$^{+}$	00.00%$^{+}$	03.33%$^{+}$
CLUSTAL W 1.83	00.00%$^{+}$	00.00%$^{+}$	00.00%$^{+}$	00.00%$^{+}$

Tab. 3.6: Percentage of sequence families where DIALIGN-T is outperformed on IRMBASE 1.0 by other methods according to the *column score*. Notation is as in Table 3.5.

Method	ref1	ref2	ref3	ref4	ref5	Total
DIALIGN 2.2	28.05%$^{+}$	21.74%$^{+}$	16.67%$^{+}$	16.67%^{0}	41.67%^{0}	26.24%$^{+}$
T-COFFEE 1.37	58.54%$^{-}$	86.96%$^{-}$	75.00%$^{-}$	25.00%^{0}	50.00%^{0}	60.99%$^{-}$
PROBCONS 1.09	71.95%$^{-}$	82.61%$^{-}$	100.00%$^{-}$	33.33%^{0}	75.00%$^{-}$	80.14%$^{-}$
POA V2	20.73%$^{+}$	34.78%$^{+}$	16.67%$^{+}$	33.33%^{0}	0.00%$^{+}$	21.99%$^{+}$
MUSCLE 3.5	71.95%$^{-}$	73.91%$^{-}$	83.33%$^{-}$	25.00%^{0}	75.00%$^{-}$	69.50%$^{-}$
CLUSTAL W 1.83	53.66%$^{-}$	56.52%^{0}	58.33%^{0}	16.67%^{0}	8.33%$^{+}$	47.52%^{0}

Tab. 3.7: Percentage of sequence families where DIALIGN-T is outperformed on BAliBASE 2.1 by other methods according to the *sum-of-pairs score*. Notation is as in Table 3.5.

Method	ref1	ref2	ref3	ref4	ref5	Total
DIALIGN 2.2	$26.83\%^{+}$	$13.04\%^{+}$	$16.67\%^{+}$	$16.67\%^{0}$	$50.00\%^{0}$	$24.82\%^{+}$
T-COFFEE 1.37	$56.10\%^{-}$	$73.91\%^{-}$	$66.67\%^{0}$	$25.00\%^{0}$	$50.00\%^{0}$	$56.74\%^{-}$
PROBCONS 1.09	$80.49\%^{-}$	$82.61\%^{-}$	$75.00\%^{-}$	$25.00\%^{0}$	$66.67\%^{-}$	$74.47\%^{-}$
POA V2	$20.73\%^{+}$	$26.09\%^{0}$	$08.33\%^{+}$	$16.67\%^{0}$	$00.00\%^{+}$	$18.44\%^{+}$
MUSCLE 3.5	$73.17\%^{-}$	$73.91\%^{-}$	$83.33\%^{-}$	$16.67\%^{0}$	$66.67\%^{-}$	$68.79\%^{-}$
CLUSTAL W 1.83	$52.44\%^{-}$	$69.57\%^{-}$	$50.00\%^{0}$	$16.67\%^{0}$	$08.33\%^{+}$	$48.23\%^{0}$

Tab. 3.8: Percentage of sequence families where DIALIGN-T is outperformed on BAliBASE 2.1 by other methods according to the *column score*. Notation as in Table 3.5.

tested methods, except MUSCLE and PROBCONS, on the sequence sets with larger insertions or deletions (ref4 and ref5 of BAliBASE).

Overall, the *relative* performance of the different alignment tools is similar under the two alternative evaluation criteria that we used (sum-of-pairs and column scores) – although, the *absolute* values of the column scores are, of course, lower than the sum-of-pairs scores. Maybe surprisingly, both versions of DIALIGN are superior to all other programs in our study on the locally related sequences from IRMBASE, while on the other hand, DIALIGN was outperformed by alternative methods on reference sets 4 and 5 of BAliBASE. These sequence sets are also considered locally related because they contain larger insertions and deletions than other BAliBASE sequences. The reason for this apparent discrepancy is that the ref4 and ref5 sequence sets in BAliBASE are not truly locally related, but they still show some similarity outside the conserved core blocks. In IRMBASE, by contrast, sequence similarity is strictly limited to the conserved motifs.

Since we re-implemented the DIALIGN algorithm *from scratch* and used a variety of novel program features, it is not possible to tell exactly to what extent each of these features contributed to the improved program performance. Systematic test runs with varying parameters indicate, however, that the superiority of DIALIGN-T compared to the previous program DIALGIN 2.2 on locally as well as on globaly related sequence families is mainly due to the program features explained in section 3. The improvement that we achieved with these heuristics is statistically significant. The features explained in section 4 also improved the program's accuracy, though here the improvement was not statistically significant.

Table 3.9 shows the running time for the seven programs that we tested in our study. DIALIGN-T is around 6 % slower than the previous implementation DIALIGN 2.2 on BAliBASE 2.1, but on IRMBASE, DIALIGN-T is approximately 30 % faster than DIALIGN 2.2. In DIALIGN, the CPU time for multiple align-

Method	Average runtime on IRMBASE 1.0	Average runtime on BAliBASE 2.1
DIALIGN-T	2.36	1.38
DIALIGN 2.2	3.33	1.30
T-COFFEE 1.37	27.54	7.64
PROBCONS 1.09	12.37	2.66
POA V2	1.44	0.58
MUSCLE 3.5	9.37	0.60
CLUSTAL W 1.83	1.41	0.47

Tab. 3.9: Average running time (in seconds) per multiple alignment for the 180 sequence families of IRMBASE and for 141 sequence families in BAliBASE 2.1. Program runs were performed on a Linux workstation (RedHat 8.0) with a 3.2 GHz Pentium 4 processor and 2 GB Ram.

ment is mainly spent on *pairwise* alignments that are performed before fragments are included into the multiple alignment. As explained in section 3.1.3, the runtime for pairwise alignment is roughly proportional to the number of fragments that are considered for alignment and, for sequences of length l_1 and l_2 and a maximum fragment length M, up to $l_1 \times l_2 \times M$ fragments are to be considered. In our program, DIALIGN-T, the maximum fragment length M is increased to 100 compared to 40 for the original DIALIGN program. Nevertheless, the program running time is only slightly increased for the globally related protein families from BAliBASE and considerably *decreased* for the locally conserved sequences from IRMBASE. This is due to the heuristic *stop criterion* for fragments introduced in section 3.1.3. The slowest program in our comparison was T-COFFEE, which is more than eleven times slower than DIALIGN-T on IRMBASE and more than five times slower on BAliBASE. POA was the fastest method. On BAliBASE, the program PROBCONS produces the best results in terms of alignment accuracy. The program is, however, the second slowest program after T-COFFEE on both BAliBASE and IRMBASE. MUSCLE provides so far the best tradeoff between running time and quality on globally related sequence families, but when it comes to local alignments both running time and alignment quality decrease drastically. The memory consumption of our method has been improved compared to DIALIGN 2.2.

3.4 Conclusion

Initially, it should be mentioned that the benchmarks were conducted in the year 2005 and thus already reach a few years back. In chapter 6 we present the

benchmark results under present conditions with respect to recent benchmark databases and the most recent available versions of the program whereby we observe that there is no substantial difference in the qualitative and performance results on the different benchmark database versions and alignment program versions when comparing those 'old' results with the new ones (see chapter 6). Having said that, the qualitative results in this chapter still hold for the most recent versions of the benchmark databases (i.e. BAliBASE and IRMBASE) and the different programs.

With the development of DIALIGN-T, we significantly improved the segment-based approach to multiple protein alignment on both local and global benchmark data. The new heuristics that we introduced, generally favour consistent groups of low-scoring fragments over isolated higher-scoring fragments. We thus improved the program performance on globally related sequence sets where the segment approach was previously inferior to programs such as CLUSTAL W and POA. On these data sets, our new method is significantly more accurate but only slightly slower than DIALIGN 2.2. On BAliBASE, the performance of our approach is now comparable to the popular global alignment program CLUSTAL W. For locally related protein families, DIALIGN-T performs significantly better and is also considerably faster than the previous DIALIGN 2.2, which was, till then, the best available method for locally related protein families. In addition to these improvements, it is now possible to use arbitrary user-defined substitution matrices, which was not possible for the original DIALIGN program. To further enhance the performance of our method, we use a combination of greedy and progressive strategies which has become the successor program, DIALIGN-TX; see chapter 6.

Finally, there are some general remarks on parameter tuning and program evaluation in multiple sequence alignment. As mentioned above, we identified suitable values for our parameters T and L, based on test runs with BAliBASE, and we assume that this is how the program parameters for most multiple protein aligners have been tuned during recent years. Therefore, the question has been raised whether current protein alignment programs are *overfitted* with respect to BAliBASE. Parameter overfitting is a serious problem for many bioinformatics algorithms. For example, many gene-prediction programs have a large number of parameters to adjust, so it is easy to tune these programs to perform well on a given set of training data. For such programs it is, therefore, absolutely necessary to clearly separate training data that are used for parameter tuning from test data that are used to evaluate the program. The situation is totally different in multiple alignment. Most multi-aligners have only a very small number of parameters to adjust. For our algorithm, for example, the only important parameters to tune are T and L. BAliBASE, on the other hand, comprises a large variety of test sequences for global multiple alignment. It consists of 139 sequence sets, each of which contains several core blocks, so there is a total of several hundred core blocks that are used to test alignment quality. It is absolutely impossible to tune a small number of parameters in such a way that they work well only on BAliBASE but not on other globally related protein sequences. Thus, if an alignment program performs well on BAliBASE, one can safely assume that it also works well on other globally related protein sequences, even if BAliBASE has been used to adjust its parameter values. In fact, it turned out that the parameters that we tuned on BAliBASE work well

not only for these *global* test data but also on the totally different artificial *local* test sequences from IRMBASE.

The real problem with BAliBASE is its heavy bias towards *globally* related sequence sets. This does not only refer to the selection of protein families that are included into BAliBASE. As mentioned above, many protein sequences in the current release of BAliBASE are *not* real-world protein sequences, but have been *artificially truncated* by the developers of BAliBASE in order to *make* them globally related. With these *non-realistic* global test sequences, the BAliBASE authors carried out a systematic program evaluation and – not surprisingly – found out that *global alignment programs generally performed better than local methods* [79]. The picture could have been totally different if realistic full-length proteins had been used instead of truncated sequences. To counterbalance the bias towards global test sets in BAliBASE, we created an additional benchmark data set consisting of simulated conserved domains embedded in non-related random sequences. The performance of alignment programs on artificial sequences should not be over-estimated as the design of such datasets is necessarily somewhat arbitrary. Nevertheless, our test runs on these simulated data give a rough impression of how different alignment methods perform on locally related data sets. In [30], it has been shown that DIALIGN-T is one of the best performing methods on amino acid sequences that differed only by short regions of deleted residues.

Finally, it should be mentioned that the performance of multiple-protein aligners under varying conditions is possible, for example by using, the full-length BAliBASE sequences or other benchmark databases such as SABmark [82, 83], Prefab [25] or Oxbench [64].

4. CONSTRUCTING MULTIPLE SEQUENCE ALIGNMENTS FROM PAIRWISE LOCAL SIMILARITIES

As described in chapter 2, the multiple sequence alignment problem can be tackled in various ways, but almost all successful algorithms are based upon constructing pairwise alignments in the initial step. When looking at the different algorithmic approaches in general and at the segment-based approach in particular one immediately faces two crucial sub-problems: the first one is to define an appropriate objective function, i.e. a scoring scheme that assesses the relevance of possible sequence similarities. The second problem is to design optimisation algorithms for optimal or near-optimal alignment in the sense of the underlying objective function. Here, efficiency in terms of computational complexity is very important since input data from real-world biological studies tends to be large. Most standard algorithms for multiple sequence alignment are based on an objective function that sums substitution scores for aligned amino acid residues and subtracts a penalty for each gap in an alignment [59]. Finding an optimal multiple-sequence alignment under this scoring scheme is already NP-complete as shown in subsection 2.2.2; as a result, all algorithmic approaches to multiple alignment are heuristics. Typically, they start off with pairwise alignments that can be computed very efficiently w.r.t. time. Most standard methods follow a progressive scheme by traversing a guide tree, thereby successively aligning pairs of sequences or sub-alignments by treating each sub-alignment as a sequence [40, 77, 60, 25, 19, 42]. There is strong evidence that the information contained in the initial pairwise alignments is already biologically valuable; see chapters 3, 6 and [74, 75, 60, 42]. While previously published versions of DIALIGN [74, 75, 53, 52] rely on time-efficient greedy heuristics that can lead to strongly sub-optimal alignments, our goal in this chapter and the next is to calculate optimal or near-optimal multiple alignments from pairwise local similarities in order to derive new efficient heuristics that replace the greedy assembly of fragments. Obviously, the quality of the given input similarities also influences the alignment quality to a great extent but we will not elaborate on the computation of pairwise alignments or other possible types of pairwise input data that may be useful as input for the algorithmic approaches presented here.

In this chapter we discuss the general problem of constructing multiple sequence alignments from local pairwise similarities, which is NP-complete in very simple cases. We will look at the intrinsic complexity in more detail by giving a modified version of the k-dimensional dynamic programming algorithm presented in subsection 2.2.2 that has running time $O(4^k n^2)$ and hence proves the problem of constructing the multiple sequence alignment from consistent pairwise local alignments to be *fixed parameter tractable (FPT)* in the number of sequences k

having a maximum sequence length of n. The class FPT [21] contains all problems that are computable in a time that is polynomial in the input size n and arbitrarily (mostly exponentially) large in another (hopefully small) parameter k.

Furthermore, we will present a plane-sweep approach that also solves the problem exactly, however, with a time bound of $O(2^{k(k-1)C_w} \cdot n^3 + n^4)$ where n denotes the input size and where C_w denotes the maximum horizontal spread of all conflicts, i.e. local pairwise similarities that cannot be included simultaneously in one multiple alignment. This algorithm, in contrast, has the advantage that it runs in polynomial time in the case the number of conflicts is strictly bounded, i.e. the exponential factor practically only comes into play when many simultaneous conflicts arise and thus makes the algorithm more suitable for solving the problem exactly for closely related input sequences compared to the modified dynamic programming. Additionally, in the next chapter the plane-sweep approach will motivate us to derive a more general algorithmic framework for developing systematical heuristics to solve the problem of constructing multiple sequence alignments from a given set of pairwise similarities specific to various problem domains.

4.1 The maximum fMSA-subgraph problem

In this chapter, we consider the following optimization problem. We are given a set of input sequences and a set $\mathcal{F}$ of local gap-free alignments of pairs of our input sequences; such alignments are called *fragments* or *fragment alignments*. A fragment f, therefore, consists of a pair of equal-length contiguous *substrings* or *segments* of two of the input sequences.

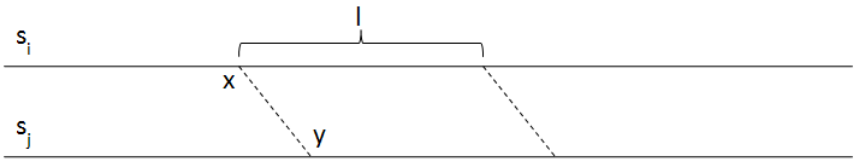

Fig. 4.1: A fragment as a gap-free alignment between two segments of length l between the sequences s_i and s_j starting at the positions x and y, respectively.

Given a weight function $w : \mathcal{F} \to \mathbb{R}_0^+$, we are looking for a *consistent* set of fragments $\mathcal{F}'$ with maximum total weight such that each $f' \in \mathcal{F}'$ is a sub-fragment of an $f \in \mathcal{F}$; sub-fragment means that f' consists of a subset of *consecutive* positions of f. Furthermore, *consistency* means that all alignments contained in $\mathcal{F}'$ can be simultaneously integrated into one single multiple alignment of the input sequences.

In this chapter, we make two additional assumptions:
Assumption 1: Firstly, we assume that any two segments belonging to fragments from the set $\mathcal{F}$ are either disjoint or coincide. In the situation of the

DIALIGN algorithms, we can fulfil this condition by appropriately chopping the fragments into multiple subfragments; by doing that we then would have to deal with an appropriate weighting of those sub-fragments since the original objective function is not linear in the sense that the weight sum of two sub-fragments f_1 and f_2 of a fragment f is usually lower than the weight of the original fragment f. Another issue that rises when chopping the fragments $\mathcal{F}$ of DIALIGN according to this assumption is that the number of fragments to consider may grow to $O(k \cdot |\mathcal{F}|)$ whereby k is the number of sequences. This is because each endpoint of an original fragment in $\mathcal{F}$ in sequence i splits at most one fragment into two sub-fragments in every pair of layers (i, j) with $1 \leq j \leq k$ and $j \neq i$. Later on, in chapter 5, we will relax this assumption using a more generalized model.

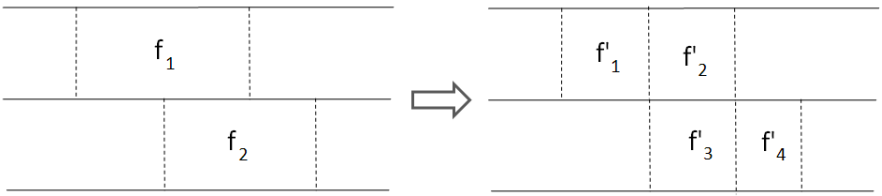

Fig. 4.2: Two fragments f_1 and f_2 that do partially overlap in the middle sequence (left) are divided into four subfragments (right) $f'_1, \ldots, f'_4$ such that the assumption that fragments either coincide or are disjoint on each sequence is being met.

We now want to associate the problem of finding an optimum subset of fragments such that all can be realized in a consistent way with a *layered graph* drawing problem.

Using the above assumption, we can identify each segment of a fragment with a node and the fragment itself as a weighted edge between those nodes having the weight score from the objective function as its edge weight. Those nodes are then placed on layers that are in a straightforward correspondence to the input sequences. This layered graph representation will help us to investigate the problem. Since our aim is to build a consistent multiple sequence alignment from a given set of input sequences, we now look at how this problem translates to the graph representation. We observe that a subset of fragments that constitutes a valid multiple sequence alignment depicts the segments of each fragment strictly overhead, i.e. each residue pair of the fragment has the same x-coordinate in the output. Hence a subset of fragments admits a valid multiple sequence alignment if and only if the corresponding subgraph of the layered graph according to the above association allows a drawing such that each edge can be drawn as a vertical straight line and the order of the nodes on each layer adheres to the order of the corresponding segments in each sequence (see Figure 4.3).

Assumption 2: Another assumption we make in our model is that for any two fragments f_1, f_2 involving the same pair of sequences, either both segments of f_1 are strictly to the left of the segments of f_2 in the respective sequences, or vice versa. In other words, we assume that the restrictions of $\mathcal{F}$ to every *pair*

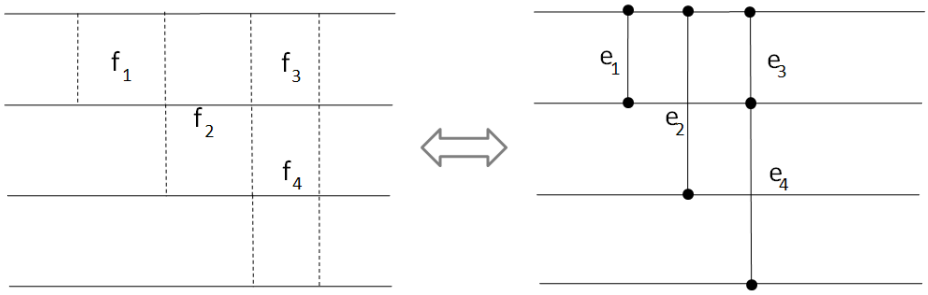

Fig. 4.3: The fragment representation of the problem (left) is translated to the graph representation such that a valid multiple sequence alignment can be obtained iff the graph representation allows a drawing such that every edge is drawn as a vertical straight line by respecting the order of the nodes on each layer as given by the fragment representation.

of input sequences is *consistent* in the sense that they all can be realized as a pairwise alignment and thus this is also called the *pairwise consistency condition or shortly PWCC*.

Later on in this and the subsequent chapter we will explain how to deal with situations when the above assumptions are relaxed.

Formally, we define the *fragment MSA instance (or shortly fMSA)* as follows:

Definition 4.1
An fMSA instance $M = (G = (V, E), k, w, L = (l_1, l_2))$ is given by:

1. a graph $G = (V, E)$
2. $k > 1$ pairwise disjoint horizontal layers in the 2-dimensional plane
3. a real-valued weight function w that assigns a weight $w(e) > 0$ to every edge $e \in E$
4. an injective function $L = (l_1, l_2)$ that assigns to each node v the two integers $L(v) = (l_1(v), l_2(v))$ whereby $1 \leq l_1(v) \leq k$ and $l_2(v) > 0$

such that the following conditions hold:

1. For each edge $e = (v, w) \in E$ the inequality $l_1(v) \neq l_1(w)$ is true.
2. For any two edges (v_1, w_1) and (v_2, w_2) with $l_1(v_1) = l_1(v_2)$, $l_1(w_1) = l_1(w_2)$ either
   (a) $l_2(v_1) < l_2(v_2)$ and $l_2(w_1) < l_2(w_2)$ or
   (b) $l_2(v_1) > l_2(v_2)$ and $l_2(w_1) > l_2(w_2)$ holds.

   The latter is also referred to as the pairwise consistency condition or PWCC.

We abbreviate the size $|M|$ of an fMSA instance by n and for all pairs of i and j between 1 and k we denote the subset of all edges that connect the nodes on layer i and layer j by $E_{i,j}$, i.e. it consists of all $e = (v, w) \in E$ with $l_1(v) = i$ and $l_1(w) = j$. The function $L = (l_1, l_2)$ can be regarded as a function that embeds each node $v \in V$ into the 2-dimensional plane by assigning a layer $l_1(v)$ and an ordering number $l_2(v)$ to all nodes on this layer.

The problem of finding a maximum consistent subset of fragments then translates to the fMSA-subgraph problem:

Definition 4.2
Given an fMSA instance $M = (G, k, w, l_1, l_2)$, the maximum fMSA-subgraph problem of M is given by: Find a subset $E' \subset E$ of edges such that the graph (V, E') can be drawn in the two-dimensional plane and the following conditions are met:

1. Any two nodes v and w with $l_1(v) = l_1(w)$ have the same y-coordinate and the x-coordinate of v is less than that of w iff $l_2(v) < l_2(w)$
2. Any two nodes v and w with $l_1(v) \neq l_1(w)$ have different y-coordinates
3. all edges are drawn as vertical straight lines.

We call such a drawing a multiple sequence drawing and we call $G_{E'} = (V, E')$ an fMSA-subgraph. If the sum

$$W(E') := \sum_{e \in E'} w(e)$$

is maximal under the above conditions we call $G_{E'}$ a maximum fMSA-subgraph.

Since all edges are drawn as vertical straight lines we immediately conclude:

Lemma 4.1
The drawing of an fMSA-subgraphs is invariant under the vertical ordering of the layers, which makes it well-defined. In the same vein, we see that for every fMSA-subgraph there is exactly one drawing modulo application of homeomorphisms and re-ordering of the layers.

4.2 NP-Completeness

Though it has been shown in [44] that finding a maximim fMSA-subgraph is already NP-complete, we will provide, using a different representation of the problem, an alternative proof and elaborate more on the computational complexity of finding maximum fMSA-subgraphs. We will show in detail that even very simple fMSA instances already carry the intrinsic complexity.

Definition 4.3
An fMSA instance $M = (G, k, w, l_1, l_2)$ is called simple if for all $e \in E$ $w(e) = 1$ and for each pair $1 \leq i, j \leq k$ there exists at most one fragment that connects sequences i and j.

The corresponding decision problem is now captured by the following definition.

Definition 4.4
The decision problem of an fMSA instance $M = (G = V, E), k, w, l_1, l_2)$ is given by the pair (M, R) where R is a positive number and the question: Is there an fMSA-subgraph E' of E of the fMSA instance M such that

$$W(E') = \sum_{e \in E'} w(e) > R.$$

We call this the 'Maximum fMSA-Subgraph Problem' or shortly MMSP.

Theorem 4.1
The Maximum fMSA-Subgraph Problem (MMSP) is NP-complete even for the class of simple fMSA instances.

As already mentioned, the above formulation of the maximum fMSA-subgraph problem for instances that do not necessarily meet the PWCC is equivalent to the maximum weight trace formulation used in [44] where it is shown that finding a maximum fMSA-subgraph is NP-complete even in the cases where every sequence contains at most two characters and there is at most one fragment between each pair of sequences. However, we will furnish a new alternative proof by reducing the problem of determining a maximum planar subgraph of the 2-layer drawing problem, where the order on the upper layer is fixed, to our problem. Eades and Whitesides [24] have shown that this problem, called 2L-MSP-1L-fix, is NP-hard even for the case in which the upper layer contains only vertices of degree one while the lower layer has vertices of degree one or two.

Proof of Theorem 4.1: Let $M = (G = (V, E), k, w, l_1, l_2)$ be an fMSA instance. By guessing a maximum subset $E' \subset E$ and using the polynomial data structure of section 3.2 and Theorem 3.1 to determine whether it forms a valid fMSA-subgraph, we see that MMSP is in NP. Thus it remains to show that finding a maximum fMSA-subgraph is also NP-hard.

We next describe how to transform a 2L-MSP-1L-fix problem instance P into an appropriate simple MMSP-problem instance P' in polynomial time and size. Thereafter we will show that a solution for P' induces an optimal solution for P and conclude that MMSP is NP-complete. The proof is given in two steps: First we allow weighted edges. Then we show how to modify the construction such that only uniformly weighted edges are present.

Let the 2L-MSP-1L-fix problem P be given by two sets $U = u_1, ...u_l$ and $B = b_1, ...b_n$ of vertices where U is ordered by x-coordinates while B is not. The edges connect vertices from U and B in a bipartite way such that the degree of all $u \in U$ is restricted to be one, while the degree of every $b \in B$ is either one or two.

1. The weighted case: For the upper layer U, we introduce a single layer L_u containing the vertices $u_1, ..., u_l$ ordered from left to right according to the order given by the 2L-MSP-1L-problem. We simply identify the names for the vertices from the same set of the two problems since the correspondence is immediate.

The representation of the lower layer B is more difficult. Every vertex $b_i \in B$ with degree 2 connected to vertices u_s and u_t is simulated by the two vertices br_i, bl_i and edges (bl_i, u_s) and (u_t, br_i) where, in this case, we assume u_s to be located to the left of u_t. Since we demand the problem to be simple, we need to place the vertices br_i, bl_i in two different unique layers l_{br_i} and l_{bl_i}. To keep bl_i left of br_i we insert a vertical dummy edge called separation edge e_i between the two new layers separating bl_i and br_i such that bl_i is situated left and br_i situated right of e_i. With the appropriate choice y for its weight, we can prevent that the two vertices from changing their order in the optimal solution, since e_i then belongs to *any* optimal solution. For nodes $b_i \in B$ with degree 1 we proceed analogously, but we omit the edge (b_r, u_t).

Now we consider any two vertices b_i and b_j from the lower layer of the original 2L-MSP-1L problem. In the MMSP, the order of b_i and b_j should not be fixed. So we consider the two pairs of layers assigned to br_i and bl_i as well as to br_j and bl_j. We insert, in total, four dummy nodes cl_{ij} to the left of bl_i, cr_{ij} to the right of br_i as well as cl_{ji} to the left of bl_j and cr_{ji} to the right of br_j. The four nodes are connected by a pair of so-called crossing edges (cl_{ij}, cr_{ji}) and (cr_{ij}, cl_{ji}). Clearly, at least one of the crossing edges is excluded in any solution of the fMSA-subgraph problem. Adjusting the weight z of the crossing edges accordingly, we can ensure that exactly one edge out of each pair of crossing edges is contained in any optimal solution of the fMSA-subgraph problem. It is obvious that each crossing edge (cl_{ij}, cr_{ji}) drawn vertically indicates the horizontal order of b_i and b_j. On the opposite site, for a fixed total order of the vertices in b, exactly one out of every pair of crossing edges can be drawn vertically.

Choosing the weights: We choose $y = (2n)^4$ and $z = (2n)^2$, where $n = |B|$, which implies $O(n) = O(|P|)$.

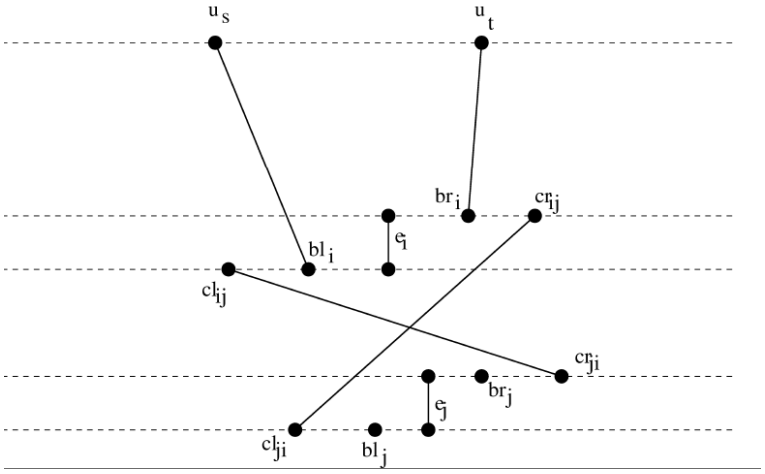

Fig. 4.4: In an optimal solution, the separation edges e_i and e_j are all realized, while for each pair of indices i, j, exactly one of the crossing edges is realized. This ensures that the optimal solution corresponds to a valid layout.

Correctness: By construction, each solution of the MMSP especially assigns x-coordinates to the vertices and so it fixes an order of the vertices br_i, bl_i for

all i.

Observation 1: All separating edges must exist in any optimal solution. Each is contributing weight $(2n)^4$ to the solution. We are not allowed to omit one of the separating edges with weight $(2n)^4$, since it would mean a larger loss than the sum of the weights of all other edges having a sum of weight bounded by $4n^3 + 2n$.

Observation 2: For any two pairs of nodes b_i and b_j, there is a conflict formed by the two crossing edges and the separating edges. By combining this with Observation 1 we see that at least one of the crossing edges must be deleted for each pair of nodes b_i and b_j and therefore a minimum of $n \cdot (n-1)/2$ crossing edges have to be removed. However, we immediately get a valid solution by fixing an arbitrary order of the b_i's and removing all edges of weight 1 (that are neither a separating edge nor a crossing edge) that have a total weight of $n < z = (2n)^2$. Such a solution would obviously embrace exactly $n \cdot (n-1)/2$ crossing edges. Combining this with the choice $z = (2n)^2$, we see that any optimal solution keeps exactly one of the crossing edges for each pair of b_i and b_j, which ensures that all pairs of nodes br_i, bl_i and br_j, bl_j are properly separated in an unique order.

Hence, we conclude that the sum of the weight of the edges is

$$n \cdot (2n)^4 + n(n-1)/2 \cdot (2n)^2 + R$$

in any optimum solution. The first term describes the n separating edges of weight $y = (2n)^4$, the second one sums the $n(n-1)/2$ remaining crossing edges, each of weight $y = (2n)^2$, and R is the number of edges of weight 1 that represent the original edges from the 2L-MSP-1L-problem and hence is bounded by $2n$. The sum of weights is thereby fixed in the first two terms for all optimum solutions and therefore any optimum solution of the MMSP with R edges of weight 1 gives us a solution of the 2L-MSP-1L-problem that is comprised of exactly the same R edges. Furthermore, by construction, any solution of the 2L-MSP-1L-problem yields an fMSA-subgraph that realizes the same edges as the original edges in the 2L-MSP-1L-problem and induces the same ordering of the nodes in B. Altogether we conclude the existence of a one-to-one correspondence of the optimum solutions of both problems, which finishes the proof of the weighted case.

2. The unweighted case: We now show how to simulate the edges with weights greater than one. Let $e = (v, w)$ be an edge of weight W where $v \in L$ and $w \in L'$ in the fMSA instance constructed above. For this edge, we introduce $W - 1$ extra layers L_i for all $1 \leq i \leq W$. Then we insert vertices v_i on every layer L_i and connect v_i to v_j with a unit-weight edge if $j > i$. Furthermore, we insert edges (v, v_i) and (w, v_i) for all i, also with weight 1. We call this construction a bundle, which replaces the original edge e.

Note that the bundle contains $W(W+1)/2$ edges and the degree of each edge within the bundle subgraph is W. To destroy an alignment of vertices from the same bundle means to destroy the edge connectivity of the bundle, which means the removal of at least W edges. This property now has the same effect as the weight W on the original single edge.

There are two kinds of edges that yield bundles: the crossing and the separating edges in the fMSA instance. We replace all those edges by bundles of corresponding sizes according to the above construction, whereby we now choose a different weight $y = (2n)^7$ for the separation edges to compensate for the much higher overall number of edges.

As before, a solution of the simple MMSP problem keeps all the bundles of the separation edges complete, since destruction of an alignment would mean a loss of weight $(2n)^7$, which is more than the total weight of all crossing bundles and 'ordinary' edges. Analogous to the above, we argue that from each pair of crossing bundles, exactly one bundle remains complete and the other is destroyed by the removal of $(2n)^2$ edges.

Hence the optimal solution for the simple MMSP problem with uniform weights has total weight of

$$\begin{aligned} n \cdot (2n)^7((2n)^7 + 1)/2 + n(n-1)/2 \cdot (2n)^2((2n)^2 + 1)/2 \\ +n(n-1)/2 \cdot ((2n)^2((2n)^2 + 1)/2 \\ -(2n)^2) \\ +R. \end{aligned}$$

The first term denotes the total weight of the separation bundles, the second is the total weight of the non-destroyed crossing bundles and the third is the total weight of the destroyed crossing bundles. Furthermore, as before, R is the number of edges of weight 1 that represent the original edges from the 2L-MSP-1L-problem P with bottom layer B. The sum of the first three terms is fixed for any problem of size $|B| = O(|P|)$ and therefore the R 'ordinary' edges directly describe a planar solution of the 2L-MSP-1L-problem and hence the maximization of its total weight. Analogous to the weighted case, we observe a one-to-one correspondence between the optimum solutions of both problems, which concludes the construction and finalizes the NP-completeness proof.

4.3 Efficient dynamic programming

We recall from subsection 2.2.2 the k-dimensional dynamic programming algorithm for solving the general multiple sequence alignment problem which has running time $O(2^k n^k)$ and is based on a k-dimensional matrix of size n^k, whereby n denotes the length of the longest sequence. We can easily use this algorithm for our optimization problem of finding an optimum alignment of a given fMSA instance $M = (V, E, k, w, l_1, l_2)$ as follows: We again identify each layer with a sequence and each node as a *unique* single character in this sequence ordered by l_2. The similarity score for each pair of characters is then given by the weight function w if there exists a corresponding edge or $-\infty$ otherwise.

Since for all $1 \leq i, j \leq k$ with $i \neq j$ the set $E_{i,j}$ of edges between layer i and j comes from a valid pairwise alignment (due to the pairwise consistency condition) we have for any position x in sequence i and any other sequence $j \neq i$ at most one edge in E that contains this node, whereas in the general multiple sequence alignment problem there may be n such edges. We will now exploit this fact to optimize the dynamic programming algorithm from section 2.2.2.

Therefore let M initially be the k-dimensional similarity matrix that stores the best known solution in $M(p_1, \ldots, p_k)$ up to position p_i in sequence i for all $1 \leq i \leq k$, whereby we start counting from 0 onwards and position 0 is associated with the initial gap symbol. We modify the algorithm such that we first go completely through sequence 1 from left to right, then through sequence 2, etc. until we are finished with sequence k. Let us assume we have gone through the first $l-1$ sequences and are now ready to process position p of sequence l. Due to the PWCC, there are at most 2^{l-1} *action points*, which means that when we compare this position with all other entries in the matrix, which has dimension $l-1$ at this processing step, we only have to consider those entries in the matrix that are associated with an edge connected to our actual position p in sequence l. The PWCC, which demands that all edges for each pair of input sequences form a consistent alignment, restricts the number of edges/fragments that position p in sequence l belongs to, to at most one for each other sequence.

Since there is maximum one such edge for every of the $l-1$ sequences already processed, we get at most 2^{l-1} entries in the matrix that are relevant to process, i.e. at most 2^{l-1} *action points*. Altogether we then have maximum $n2^{l-1}$ action points for sequence l and the overall sum of action points is then

$$\sum_{1 \leq l \leq k} n2^{l-1} = n(2^k - 1) = O(2^k n).$$

We will now explain how we can restrict the size of the matrix from $O(n^k)$, in general, to the number of actions points $O(2^k n)$. Therefore, we initially change the index range for M from $\{1, \ldots, n\}^k$ to $\{1, \ldots, n^k\}$ whereby the index k-tuple $(p_1, \ldots, p_k)$ is translated *uniquely* to the new index

$$\sum_{1 \leq i \leq k} n^i p_i$$

and the entry of the original matrix $M(p_1, \ldots, p_k)$ can be now found in

$$M(\sum_{1 \leq i \leq k} n^i p_i).$$

From the way we traverse the matrix starting with sequence 1 from left to right, etc. up to sequence l from left to right the natural ordering of the numbers $1, \ldots, n^k$ is exactly the same as the processing order of the respective positions in the similarity matrix M. Let a and b be two action points with $1 \leq a < b \leq n^k$ such that there is no other action point between a and b. We now arrange the action points as a double-linked list L whereby each list element of action point a carries the similarity matrix entry $M(a)$ as payload. Thus we can omit the overall storage of M by using L instead and we can thereby reduce the storage size down to $O(2^k n)$. When we now come to position p in sequence l we can immediately determine the new maximum 2^{l-1} action points determined by its outbound edges to sequences $1, \ldots, l-1$. We then sort those action points in time $O((l-1)2^{l-1})$ and add them one by one from left to right to the linked list L. In the original dynamic programming algorithm, we immediately have access to the relevant 2^k predecessor entries in M that are relevant to process position p in sequence l, which is not the case with our linked list. Hence we

have to seek those entries in the linked list by traversing it once from the start to the end and updating a data structure holding the relevant 2^k predecessor entries imposing an additional time of

$$O\left(2^k + \sum_{1 \leq t \leq l} n2^{t-1}\right) = O\left(2^k + n(2^l - 1)\right) = O\left(2^k + 2^l n\right).$$

Assuming we have $n > k$ we get an overall running time for the algorithm of

$$\begin{aligned} & O\left(\sum_{1 \leq l \leq k} \left(n2^{l-1} \cdot ((l-1)2^{l-1} + 2^k + 2^l n)\right)\right) \\ \leq O\left((k-1)2^k n \cdot \sum_{1 \leq l \leq k} \left(2^{l-1}\right)\right) + O\left(n^2 \cdot \sum_{1 \leq l \leq k} \left(4^{l-1}\right)\right) & \\ \leq O\left(k2^k n \cdot 2^k + n^2 4^k\right) & \\ \leq O\left(4^k \cdot n^2\right). & \end{aligned}$$

Theorem 4.2
The modified k-dimensional dynamic programming algorithm described above has an overall running time of $O\left(4^k \cdot n^2\right)$ for an fMSA instance $M = (G, k, w, l_1, l_2)$ with a maximum sequence length $n > k$. Moreover, the problem of finding a maximum fMSA-subgraph is fixed parameter tractable (FPT) with parameter k.

In distantly related sequence pairs it may be unclear how the correct pairwise alignment looks and maybe multiple competing pairwise alignments of a sequence pair may occur. In such a situation, one wants all or some of those competing pairwise alignments to be considered when assembling the multiple sequence alignment, however, the PWCC would then clearly not be met. Since this is still an interesting case, we now investigate the complexity in the case when the pairwise consistency condition does not necessarily hold and also determine which additional parameter carries the intrinsic complexity in such situations.

Definition 4.5
Let $m > 0$ be such that for any sequence pair $1 \leq i < j \leq k$ and any position p in sequence i the number of edges between sequence i and j that have one endpoint in p is bounded by m. We then call m the *maximum pairwise segment spread (MPSS)*.

Of course, in fMSA instances that meet the PWCC, the maximum pairwise segment spread m is always $m = 1$ and in the general case it is always maximum n, whereby n is again the length of the longest sequence. We will now look at the impact of the modified dynamic programming algorithm when we have to deal with a situation where the PWCC does not necessarily hold und we may have $m > 1$. For that, let us again assume that we have gone through the first $l - 1$ sequences and are about to process position p of sequence l in our modified dynamic programming algorithm; then we now have at most $(m+1)^{l-1}$

Fig. 4.5: If the PWCC holds, position p can be part of at most one edge to each other sequence (left). Generally however there can be arbitrary many edges from p to each other sequence. The maximum possible number of such edges is called the maximum pairwise segment spread (MPSS) and in this example (right) it is bounded by 5.

action points occurring, which analogously yields an overall running time for the algorithm of

$$O\left(\sum_{1\leq l\leq k}\left(n(m+1)^{l-1}\cdot((l-1)(m+1)^{l-1}\log(m+1)+(m+1)^k+(m+1)^l n)\right)\right)$$

$$\leq O\left((k\frac{\log(m+1)+1}{m+1}(m+1)^k n\cdot\sum_{1\leq l\leq k}(m+1)^l\right)+O\left(n^2\cdot\sum_{1\leq l\leq k}(m+1)^{2l-1}\right)$$

$$\leq O\left((k(m+1)^k n\cdot(m+1)^{k+1}+n^2(m+1)^{2k+1}\right)$$

$$\leq O\left((m+1)^{2k+1}\cdot n^2\right)$$

(the $\log(m+1)$ comes from sorting the action points for each position p).

We therefore conclude the following theorem:

Theorem 4.3
The problem of finding a maximum fMSA-subgraph of a given fMSA instance $M=(G,k,w,l_1,l_2)$ when the PWCC does not hold is FPT in the number of sequences k and the maximum pairwise sequence spread plus 1, i.e. $m+1$, and has a running time of $O\left((m+1)^{2k+1}\cdot n^2\right)$.

4.4 The plane-sweep approach

The running time of the modified k-dimensional dynamic programming algorithm of the previous section is invariant under the number of conflicting situations, i.e. a collection of fragments that cannot all be realized in parallel without violating the consistency. Thus it is not useful for exactly solving large real-world inputs, especially in cases where the input sequences do not give rise to many parallely conflicting situations, e.g. for rather long but few input sequences. To compensate for this shortfall, we will give an alternative algorithm based on a plane-sweep approach and which leads to high computational complexity *only* if the number of parallely conflicting situations becomes very high.

Roughly, the algorithm is based upon a plane-sweep approach adding the edges in a left-to-right manner using an appropriate sorting of the edges whereby it

keeps a set $\mathcal{S}$ of possible candidate solutions and tries to incorporate the next edge in each of the already existing candidates in $\mathcal{S}$. This incorporation possibly involves some splits of candidates into a set of alternatives which is healed by a subsequent consolidation step that reduces the pool of candidates without losing all options for an optimum solution.

4.4.1 Conflicting cycles

Conflicting cycles pose the major obstacle to valid subgraphs and hence they are required to be looked at in more detail. Briefly, a conflicting cycle is a minimum set of edges that can not all be realized in parallel, but when removing only one, the remainder becomes a valid fMSA-subgraph. More precisely:

Definition 4.6
Let $C \subset E$ of a fMSA instance $M = (G = (V, E), k, w, l_1, l_2)$ and for all $1 \leq i, j \leq k$ we set $C_{i,j} = C \cap E_{i,j}$. We call C a conflicting cycle iff all of the following conditions hold:

1. The size of all $C_{i,j}$ is either 0 or 1.
2. All $C \backslash C_{i,j}$ with $C_{i,j} \neq \emptyset$ yield a maximal fMSA-subgraph of the fMSA instance induced by $C \subset E$.

The set of all conflicting cycles of a given fMSA instance M is denoted by $\mathcal{C}(M)$.

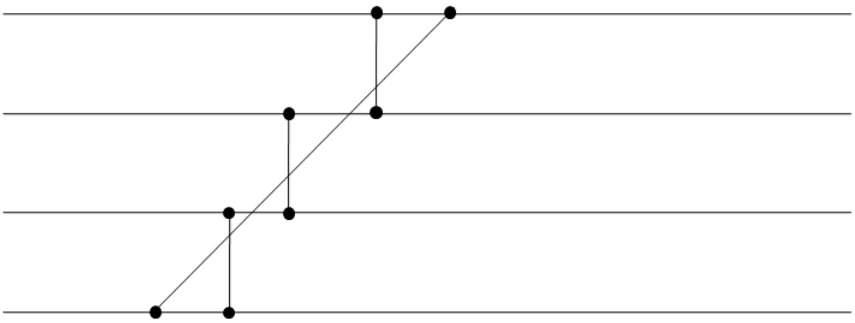

Fig. 4.6: Depiction of a conflicting cycle between four sequences and involving four edges.

Lemma 4.2
Let $\mathcal{C} := \mathcal{C}(M)$ be the set of all conflicting cycles of an fMSA instance M. Any $F \subset E$ induces an fMSA-subgraph $G_{E \backslash F} = (V, E \backslash F)$ of M if and only if for every $C \in \mathcal{C}$ there exists a pair (i, j) with $C_{i,j} \neq \emptyset$ and $C_{i,j} \subset F$.

Proof: By condition 2 of definition 4.6, any F that induces an fMSA-subgraph has to destroy every conflicting cycle. Hence, we assume we have an $F \subset E$ that

destroys all conflicting cycles in $\mathcal{C}$, but there is at least one $e = (u, v) \in E \backslash F$ that cannot be drawn vertically with a maximal subset of $U = E \backslash F$ as per the definition of fMSA-subgraphs. Then there is a chain of edges $f_1, \ldots, f_l \in U$ such that f_1 has one endpoint in the same layer as v and f_l one endpoint in the same layer as w such that $E \cup \{e\}$ fulfills all four requirements for a conflicting cycle as per definition 4.6. The existence of such a chain is obvious since otherwise there would be no alternative connection between layer $l_1(v)$ and $l_1(w)$ that crosses e and thus would prevent e being drawn vertically. Altogether, the existence of U finishes the proof.

One important ingredient for obtaining proper bounds of the plane-sweep algorithm is the *cycle-width*, which we will now define in our context.

Definition 4.7
Let M be any fMSA instance and let C be any conflicting cycle of M. Then the cycle-width w of C is defined as follows:

$$c_w(C) = \max_{(r,s),(t,u) \in C} \{|l_2(r) - l_2(u)|, |l_2(s) - l_2(t)|\}.$$

The max-cycle-width C_w of M is then defined to be the maximum among all cycle-widths of conflicting cycles occurring in M, i.e.

$$C_w(M) = \max_{C \in \mathcal{C}} c_w(C).$$

The cycle-width describes the maximum l_2-difference of any two nodes occurring in a conflicting cycle and thus gives an intuitive measurement of how far the cycle spreads horizontally. Later we will discover the importance of the max-cycle-width in more detail when we investigate the complexity of the plane-sweep algorithm.

The handling of conflicting cycles in the algorithm requires an auxiliary graph $G_a = (V_a, E_a)$ that contains two distinct nodes u_e, v_e for every edge $e = (u, v) \in E$. For any two edges $e = (u, v)$ and $f = (p, q)$ that meet all of the following conditions we add a directed edge (u_e, p_f) to E_a.

- $l_1(u) \neq l_1(q)$
- $l_1(v) = l_1(p)$
- $l_2(v) \leq l_2(p)$

Lemma 4.3
There is a 1-to-1 correspondence between the conflicting cycles of M and the directed cycles in G_a. The auxiliary graph G_a can be constructed in $O(n)$ time.

Proof: This is quite straightforward since the edges connecting nodes of a directed cycle of G_a immediately result in a conflicting cycle and vice versa.

If we want to know whether two edges $e = (u, v)$ and $f = (p, q)$ in E are part of a common conflicting cycle C, we perform depth-first searches to find directed paths in G_a from each of the nodes u_e, v_e to each of the nodes p_f, t_f and backwards which makes 8 searches in total. We then try to combine the

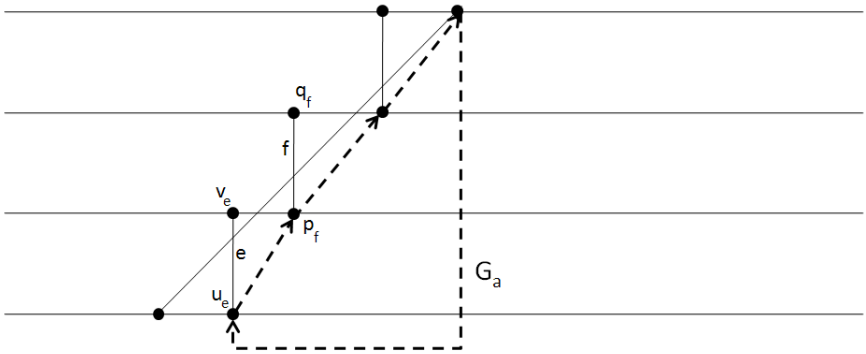

Fig. 4.7: Construction of the auxiliary graph G_a whereby each conflicting cycle corresponds to a directed cycle in the auxiliary graph G_a.

resulting paths to all possible directed cycles containing one of the nodes u_e, v_e *and* one of the nodes p_f, t_f. Finally e and f are part of a common conflicting cycle *if and only if* at least one could be realized, which concludes the proof.

Lemma 4.4
The cycle-width C_w of any fMSA instance M can be determined in time $O(n^3)$.

Proof: This is obvious by using a depth-first search in the auxiliary graph G_a for each pair of edges $e, f \in E$ to find out if they are part of a common conflicting cycle and if so, we compute all l_2 differences of all four relevant node combinations of e and f.

4.4.2 The index function

Another instrument we need is the index function that assigns an index value to every candidate solution and whenever two candidates obtain the same index we will only keep the one with the higher weight sum thus allowing us to control the computational complexity. As already mentioned, the algorithm considers one edge of E after the other in a left-to-right manner. In order to keep track of which edges have already been processed we associate a flag 'seen(e)' with each edge $e \in E$ which equals 1 iff e has already been processed. Additionally, we maintain a pool $\mathcal{S}$ of possible candidate solutions and introduce a bit-vector valued index function $\text{ind}(\cdot) : \mathcal{S} \to \{0,1\}^r$ on $\mathcal{S}$ which we will define shortly. The number r varies from step to step and therefore depends on $\mathcal{S}$. We regard E to be subdivided into $E_1 \subset E$ and $E_0 \subset E$, where E_1 is comprised of all edges with $\text{seen}(e) = 1$ and E_0 embraces all edges with $\text{seen}(e) = 0$. At any stage in the algorithm we define $R_{i,j}$ to be the subset of edges $e \in E_1 \cap E_{i,j}$ such that there exists a conflicting cycle C_e with $C_e \cap E_0 \neq \emptyset$ and $e \in C_e$. The size of the index is given by

$$r := \sum_{1 \leq i < j \leq k} |R_{i,j}|$$

and we fix a lexicographical ordering on the set of all pairs (i, j) with $1 \leq i, j \leq k$. Each set $E_{i,j}$, and thereby also each $R_{i,j}$, is naturally ordered by the function

l_2 due to the pairwise consistency condition (PWCC): let $e = (u, v) \in E_{i,j}$ and $f = (p, q) = E_{i,j}$ such that u and p are on layer i, $l_1(u) = l_1(p) = i$, then the PWCC demands $l_2(u) < l_2(v)$ iff $l_2(v) < l_2(q)$.

To facilitate the understanding of the index function, we interpret ind$(\cdot)$ to be a function having values in blocks of bit-valued vectors as follows:

$$\{0,1\}^{|R_{1,2}|} \times \{0,1\}^{|R_{1,3}|} \times \ldots \times \{0,1\}^{|R_{k-1,k}|}.$$

Then the index function is defined such that for any $S \in \mathcal{S}$, the l-th bit in the (i, j)-th block of ind(S) is set to 1 iff the l-th edge of $R_{i,j}$ (w.r.t the ordering of $R_{i,j}$ given by H_M) belongs to S; otherwise it is set to 0.

4.4.3 Description of the plane-sweep algorithm

The previous subsections endow us with the necessary preparations to now formulate the final algorithm. Initially, we start with an empty set of solution candidates $\mathcal{S} = \emptyset$ and also compute the auxiliary graph G_a. Furthermore, we initialize all $R_{i,j} := \emptyset$. Then, the algorithm first considers all edges (v, w) with $l_2(v) = 1$ or $l_2(w) = 1$ and thereafter all edges with $l_2(v) = 2$ or $l_2(w) = 2$ and so on. We track this behaviour by introducing the progress number l which is initially set to 1 and incremented whenever all edges with l_2 number equal to l in one of their endpoints have been consumed. The iterative processing of the edges is comprised of three stages as long as there are unprocessed edges $e \in E$ with seen$(e) = 0$ available. After each iteration the set of candidate solutions $\mathcal{S}$ can be uniquely enumerated by the index function and thus has a maximum size of 2^r whereby r denotes the current index size. The three stages for each iteration are as follows:

1. Choose: In the case where there is no unseen edge available we finish the algorithm. As long as there is no unseen edge (v, w) with $l_2(v) = l$ or $l_2(w) = l$, we increase the progress number l by 1 and finally choose an arbitrary edge $e = (v, w)$ with $l_2(v) = l$ or $l_2(w) = l$.
2. Split: Let e be the edge chosen in the previous stage and let $1 \leq i, j \leq k$ be the pair of integers such that $e \in E_{i,j}$. In case e is the very first edge to be processed we create the solution candidate $S := \{e\}$, add it to $\mathcal{S}$ and proceed to the next stage. In all other cases we iterate through the set $\mathcal{S}$ and for each $S \in \mathcal{S}$ we determine the set of $P_{S,e}$ of *split-solutions* according to e as follows: If e can be added to S without violating the fMSA-subgraph conditions we set

$$P_{S,e} := \{S \cup \{e\}\};$$

otherwise we start with $P_{S,e} = \{S\}$ and for each subset $S' \subset S$ such that $S'_e := S' \cup \{e\}$ defines a valid fMSA-subgraph and such that S' realizes exactly the same edges that do not belong to the index as S we add S'_e to $P_{S,e}$. We compute $P_{S,e}$ by iterating through all 2^r possible configurations of the index that may determine such an S' and performing linear searches in G_a restricted to $E_1 \cup \{e\}$ to find out if e closes a conflicting cycle in the candidate S', whereby r denotes the current index size.
3. Consolidate: This stage starts with setting seen$(e) := 1$ and

$$\mathcal{P}_e := \bigcup_{S \in \mathcal{S}} P_{S,e}.$$

Thereafter we increase the index by adding e to it and by adjusting all $R_{i,j}$. The latter adjustment can only reduce or maintain the sizes of each $R_{i,j}$ and is necessary since edge e, which is now processed, has possibly closed the last conflicting cycle of edges belonging to some of the $R_{i,j}$'s. We then compute the new index for all $P \in \mathcal{P}_e$ and for any two $P_1, P_2 \in \mathcal{P}_e$ with the same index and we erase the one with the lower weight, $W(P_1)$ or $W(P_2)$. Then the remaining solutions form the new $\mathcal{S}$ as the result of the current iteration.

After all edges have been processed, we choose any solution $S \in \mathcal{S}$ realizing the maximum weight $W(S)$ in $\mathcal{S}$ to be the output solution.

We directly get the following upper bound of the algorithm's running time:

Lemma 4.5
The plane-sweep algorithm has running time

$$O\left(4^{r_{max}} \cdot n^3 + n^4\right)$$

where r_{max} denotes the maximum index size of r throughout all iterations.

Proof: Initially, by Lemma 4.3, the auxiliary graph G_a can be constructed in time $O(n)$. Hence we now have to look at the n iterations of the three steps 'Choose', 'Split' and 'Consolidate', whereby the 'Choose' step can easily be done in $O(n)$. The split step requires time $O(2^r \cdot 2^r \cdot n^2)$ to compute for maximum 2^r candidates in $\mathcal{S}$ the set $P_{S,e}$, each in time $O(2^r \cdot n^2)$. Regarding the 'Consolidate' step, the computation of $\mathcal{S}$ and the weights of all its candidates can be done in time $O(2^r \cdot 2^r \cdot n)$ since

$$|\mathcal{P}_e| \leq 2^r \cdot 2^r$$

and by again using the auxiliary graph G_a the update of all $R_{i,j}$ can be achieved in time $O(n^3)$. Hence for the 'Consolidate' step we need time $O(2^{2r_{max}} \cdot n + n^3)$. Altogether we iterate over $O(n)$ edges which yields an overall computational complexity of $O\left(4^{r_{max}} \cdot n^3 + n^4\right)$ which completes the proof.

Although the algorithm is naturally only sensitive to the number of simultaneously occuring conflicting cycles, we may end up with $O(4^{r_{max}}) = O(4^n)$ in the worst case. However, we will later see how this upper bound can be significantly reduced using the max-cycle-width.

4.4.4 Correctness

Before we elaborate more on the algorithm's performance, we show that the plane-sweep algorithm indeed finds a maximum fMSA-subgraph and therefore fulfills its intended purpose. For this proof we need the definition of the following invariant.

Definition 4.8
The following is defined as the 'Plane-Sweep Invariant':
PSI: There exists a candidate $S' \in \mathcal{S}$, a subset $E'_1 \subset S'$ and a subset $E'_0 \subset E_0$ such that

$$S' \backslash E'_1 \cup E'_0$$

is a maximum fMSA-subgraph of the fMSA instance M. In this case we say that S' is extensible to a maximum fMSA-subgraph of M.

Using this definition it is sufficient to show the following:

Lemma 4.6
The invariant PSI remains true after each iteration that processes the next edge e in the plane-sweep algorithm presented above.

Proof: Before any edge has been considered PSI is trivially true. Therefore, by induction, we have to show that if PSI is true and a new edge $e \in E$ is processed PSI is not violated. Hence, we now assume PSI holds and accordingly we choose S' which is extensible to a maximum fMSA-subgraph. Therefore, let $E'_1 \subset S'$ and $E'_0 \subset E_0$ such that $S' \backslash E'_1 \cup E'_0$ is a maximum fMSA-subgraph of M according to PSI. We are now going to show that PSI remains valid after having added the next edge $e \in E$. According to Lemma 4.2, every edge of E'_1 must have a conflicting cycle in common with at least one edge in E'_0 since otherwise such an edge could be added to $S' \backslash E'_1 \cup E'_0$ giving us a solution candidate with even higher weight, which contradicts PSI. We recall that for every pair $1 \leq i < j \leq k$ the set $R_{i,j}$ is only reduced 'from left' by a subset $\hat{R}_{i,j} \subset R_{i,j}$ iff there is no unseen edge f such that there is a conflicting cycle embracing f and edges in $\hat{R}_{i,j}$. Hence we conclude that E'_1 only contains edges that are part of the index R, which is comprised of all $R_{i,j}$ with $1 \leq i < j \leq k$.

We now have to distinguish two cases: Either e fits into S' without removing edges from S' or it does not.
Case 1: Assume $S' \cup \{e\}$ forms a valid alignment. The 'Consolidate' step can replace the candidate $S' \cup \{e\}$ only by a candidate S'' which has at least the same weight sum and differs from $S' \cup \{e\}$ only in edges that are *not* a part of a yet unclosed conflicting cycle (since by its choice S'' can replace only candidates with the same index). If $e \notin E'_0$, this means that S'' and $S' \cup \{e\}$ differ only in edges $E \backslash E'_1$ and $S'' \backslash (E'_1 \cup \{e\}) \cup E'_0$ or $S'' \backslash E'_1 \cup E'_0$ forms a maximum fMSA-subgraph depending on whether $e \in S''$ or not. Otherwise, if $e \in E'_0$, the candidate $S'' \backslash E'_1 \cup (E'_0 \backslash \{e\})$ would be a maximum fMSA-subgraph for the same reason that the edges in $E'_0 \backslash \{e\}$ do not have a conflicting cycle in common with edges in which S'' and $S' \cup \{e\}$ have differences.
Case 2: Assume $S' \cup \{e\}$ does not form a valid alignment. If $e \notin E'_0$, S' can again be replaced in the 'Consolidate' step only by another candidate S'' with the same weight as S' and differing only in edges that do not have a conflicting cycle in common with edges in E'_0 and thus we obtain a maximum fMSA-subgraph by $S'' \backslash E'_1 \cup E'_0$. Otherwise, if $e \in E'_0$, the 'Split' step generates a candidate $S'' \in P_{S',e}$ such that $S'' \backslash E''_1 \cup (E'_0 \backslash \{e\})$ gives us a maximum fMSA-subgraph. The subsequent 'Consolidate' step may replace S'' by S''', however, it must again have at least the same weight as S'' and differ only in edges not belonging to E''_1 such that $S''' \backslash E''_1 \cup (E'_0 \backslash \{e\})$ would be a maximum fMSA-subgraph.

Altogether we have shown PSI remains true after having processed e and by induction it is therefore true across all iteration steps, which completes the proof.

4.4.5 Performance investigation

The main computational complexity comes from the index size during the iterations of the three stages in the algorithm. At the moment it is not clear if the maximum index size r_{max} can be even $\Omega(n)$, which would yield a running time exponential in n. The following theorem explains how we the computational complexity can be properly bounded by giving more detailed insight into the intrinsic complexity of the problem.

Theorem 4.4
Let $M = (G = (V, E), k, w, l_1, l_2)$ be an fMSA instance with k layers and max-cycle-width $C_w := C_w(M)$; then the plane-sweep algorithm is an FPT-algorithm in parameters k and C_w and has a running time of $O(2^{k(k-1)C_w} \cdot n^3 + n^4)$.

Proof: From Lemma 4.5 we conclude that the running time of the plane-sweep algorithm is bounded by $O(4^{r_{max}} \cdot n^3 + n^4)$ whereby r_{max} is the maximum index - in other words, it is the maximum of

$$|R| = \sum_{1 \leq i < j \leq k} |R_{i,j}|$$

across all iterations. Hence it is sufficient to show that at any stage of the algorithm, $R_{i,j}$ is bounded by C_w for all $1 \leq i < j \leq k$ since then the index would be bounded by

$$|R| \leq \sum_{1 \leq i < j \leq k} C_w = \frac{k(k-1)}{2} C_w.$$

Because of the definition of the pairwise consistency condition that the edges between each pair of layers form a valid fMSA-subgraph we can choose the rightmost edge $e' = (u, v)$ of $R_{i,j}$ that, by definition, is comprised of all seen edges in $E_{i,j}$ and that belong to uncompleted conflicting cycles. Assume the current progress number is l. By the 'Choose' step either $l_2(u) \leq l$ or $l_2(v) \leq l$ and without loss of generality we can therefore assume for e' and every other edge in $R_{i,j}$ that its endpoint in layer i has an l_2-value less than or equal to l.

Again due to the pairwise consistency condition it is prohibited that any two different edges in $E_{i,j}$ share a common edge in layer i or layer j and therefore their endpoints always have a different l_2-number in each layer. Hence we can assign a unique number $l_{2,i}(e)$ to all edges in $e \in R_{i,j}$ that is given by the l_2-number of the node belonging to layer i. We recall that $R_{i,j}$ is defined to contain only edges that are part of uncompleted conflicting cycles and that the definition of the maximum cycle-width C_w describes the maximum l_2-difference of any two nodes in any conflicting cycle. This especially means that for every edge $e \in R_{i,j}$, all conflicting cycles that e is part of will be fully considered at latest when the progress number $l_{2,i}(e) + C_w$ has been completed, giving us, for the current progress number l, the inequality $l_{2,i}(e) + C_w \geq l$ (otherwise, e would not be part of the index). Since we have chosen layer i to have l_2 numbers for $R_{i,j}$ less than the current progress number l, we conclude for each edge $e \in R_{i,j}$

$$l \geq l_{2,i}(e) \geq l - C_w.$$

Thus the *unique* $l_{2,i}$ number for each edge in $R_{i,j}$ is bounded in an interval of size C_w and therefore the size of $R_{i,j}$ is at most C_w. Since there are $\frac{k\cdot(k-1)}{2}$ pairs of layers, the maximum index size r_{max} is thus bounded by $\frac{k(k-1)C_w}{2}$. Using Lemma 4.5 we obtain for the overall running time

$$O\left(4^{r_{max}} \cdot n^3 + n^4\right) \leq O(2^{k(k-1)C_w} \cdot n^3 + n^4),$$

which completes the proof.

4.4.6 Relaxing the pairwise consistency condition

Until now, we have only investigated cases where the input fMSA instance meets the pairwise consistency condition (PWCC), which means, in other words, that for each pair of sequences all input similarities form a valid pairwise alignment. In some situations when the sequences are very distantly related, there may exist multiple competing alternatives for the pairwise alignment and thus it may be helpful to consider some or all alternatives in parallel by relaxing the PWCC by allowing these competing alternatives to occur in the input. In such cases the plane-sweep algorithm will definitely deliver the correct result since it doesn't exploit the PWCC, but the impact on the running time of the algorithm is obviously not within the same bounds given in the previous section. We will therefore elaborate in the following on how to alter the plane-sweep algorithm and on controlling the computational complexity.

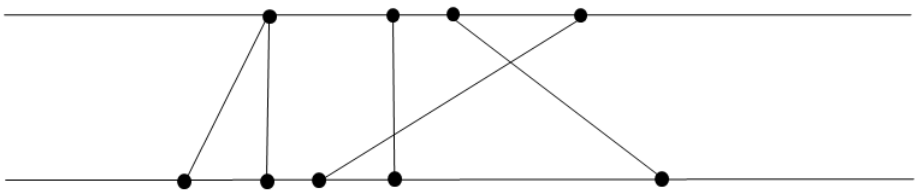

Fig. 4.8: A pair of layers where the pairwise consistency condition PWCC does not hold since not all edges can be realized simultaneously in an fMSA-subgraph.

Let us look at the maximum cycle-width C_w, which is now also determined by cycles comprised solely of two edges coming from the same pair of layers, hence, we have to expect higher values for C_w compared to the previous case where the PWCC holds. Luckily, we can modify the algorithm such that its complexity is still controlled by the number of layers k and the maximum cycle-width C_w, which of course can be $\Omega(n)$.

The issue when controlling the complexity lies in the fact that in the case where the PWCC does not necessarily hold we lack the natural l_2-ordering of the edges in $E_{i,j}$ and in turn also the natural ordering in $R_{i,j}$ being used in the proof of Theorem 4.4. Nevertheless, the endpoints of the edges in $E_{i,j}$ are still ordered on the layers i and j *separately* by l_2 and we therefore apply *two* orderings on $E_{i,j}$: one with respect to layer i and one with respect to layer j, which of course doubles the index. More precisely, we define a new ordering function l_2'

as a derivate of the l_2 function that assigns for every edge $e \in E_{i,j}$ an ordering number $l_2'(e,i)$ among all edges in $E_{i,j}$ which is given by the l_2 value of the node in layer i. Since the PWCC does not necessarily hold, this definition does not yet yield a *unique* ordering and we thus have to find a way of handling the nodes of degree > 1. This is done by using the ordering induced by their respective neighbour nodes on layer j, which gives us a valid sub-numbering for all edges that have the same l_2 value on layer i. We arrange this numbering such that all nodes of $E_{i,j}$ are uniquely numbered consecutively from 1 to $|E_{i,j}|$ on layer i by $l_2'(\cdot, i)$. Furthermore, we modify the plane-sweep algorithm by using the new function l_2' instead of l_2 in the 'Choose' step. The index function in the original algorithm is given by assigning 0 or 1 to edges of $E_{i,j}$ with $i < j$ and is denoted by $R_{i,j}$ whereby the edges of $R_{i,j}$ are ordered strictly by l_2, which is well-defined if the PWCC holds. If we now double the index by adding all $R_{j,i}$ with $i < j$ to it and change the underlying ordering function for $R_{i,j}$ and $R_{j,i}$ to $l_2'(\cdot, i)$ and $l_2'(\cdot, j)$, respectively, we immediately obtain a well-defined strict ordering for $R_{i,j}$ and $R_{j,i}$. Additionally, we alter the algorithm such that $R_{i,j}$ and $R_{j,i}$ are always amended by edges with $l_2'(\cdot, i)$- and $l_2'(\cdot, j)$-number, respectively, less than or equal to the current processing number l. All in all, by the above construction, the original $R_{i,j}$ has now been split into two subsets that are not necessarily disjoint and the overall index is now less than or equal to twice its original size. Obviously, our modifications of the plane-sweep algorithm do not affect the correctness and thus yield the following theorem.

Theorem 4.5
Let M be an fMSA instance of k layers and max-cycle-width $C_w := C_w(M)$ that does not meet PWCC, then the modified plane-sweep algorithm is an FPT-algorithm in the two parameter k and C_w having running time $O(4^{k(k-1)C_w} \cdot n^3 + n^4)$.

We conclude that by relaxing the PWCC, the plane-sweep algorithm can be modified slightly such that the computational complexity can still be bounded by an appropriate factor only depending on the number of layers k and the maximum cycle-width C_w. However, we have to expect higher values for C_w since the absence of the PWCC gives rise to more conflicting cycles, which of course are determined by how 'inconsistent' the pairwise alignments are. Especially in the extreme case when there is an edge between each pair of nodes on each pair of layers, the cycle-width C_w immediately becomes $\Omega(l)$ whereby l is the maximum number of nodes in one layer.

In the next chapter 5 we will see how we can carry the basic idea of the plane-sweep algorithm further by giving a general algorithmic framework for solving the fMSA-subgraph/subset problem heuristically or exactly.

5. THE ALGORITHMIC FRAMEWORK

In the preceding chapter we described a plane-sweep algorithm that solves the fMSA-subgraph problem exactly by being sensitive only to the number of parallely arising conflicting situations. We will now keep the idea of iterating the three steps 'Choose', 'Split' and 'Consolidate' on a set of possible solution candidates but will extend it further to a more general framework of algorithms for both, computing *maximum* MSA-subgraphs or heuristic approximations thereof. The design of the framework is very generic such that it covers the two extremes: the plane-sweep algorithm and the implementations of DIALIGN, DIALIGN-T and DIALIGN-TX (see chapter 6). Up to now we have used for the sake of simplicity, a restricted model where the fragments either coincide or are completely disjoint on each sequence, i.e. do not partially overlap. That model allowed us to use a graph representation of the assembly problem. However, in this chapter we will relax this condition and introduce the notion of the generalized fMSA-subgraph problem where fragments may overlap or even whole multiple sub-alignments may be allowed.

5.1 Framework definition

In order to give the framework a more general character we introduce the notion of a generalized fMSA instance:

Definition 5.1
Let $M = (G = (V, E), k, w, L = (l_1, l_2))$ be a given fMSA instance that may or may not be pairwise consistent and let $\mathcal{E}$ be a subdivision of E, i.e. it consists of subsets of E whereby the union of all those subsets yields back E. Furthermore, let $\hat{w}$ be an extension of w from elements of E to the set of all subsets of E, which we denote by 2^E. Then we call $(M, \mathcal{E}, \hat{w})$ a generalized fMSA instance, $\mathcal{E}$ a generalized similarity set being comprised of subsets similarities and $\hat{w}$ a generalized weight function.

A standard fMSA instance $M = (G = (V, E), k, w, l_1, l_2)$ can be associated with a generalized one in a straightforward way by defining $\mathcal{E}$ to be comprised of all $\{e\}$ for each $e \in E$ and setting $\hat{w} = W$, whereby for each $E' \subset E$ and the

$$W(E') := \sum_{e \in E'} w(e).$$

In the previous chapter the restricted model of fMSA instances only allowed fragments that do not partially overlap on the sequences and we thus had to

chop a fragment f into appropriate sub-fragments $f_1, \ldots, f_i$ associated with edges $e_1, \ldots, e_i$ such that they fit into that restricted model. In the generalized setup we can combine those chopped sub-fragments back into their original fragment by adding the subset $E_f = \{e_1, \ldots, e_i\}$ to $\mathcal{E}$ and setting $\hat{w}(E_f) := w(f)$, whereby w is the original objective function on the set of fragments. However, the generalized model even allows whole multiple sub-alignments with individual weights that may or may not account for gaps, thus allowing a wide variety of inputs from simple fragments up to anchor points and conserved motifs a priori known.

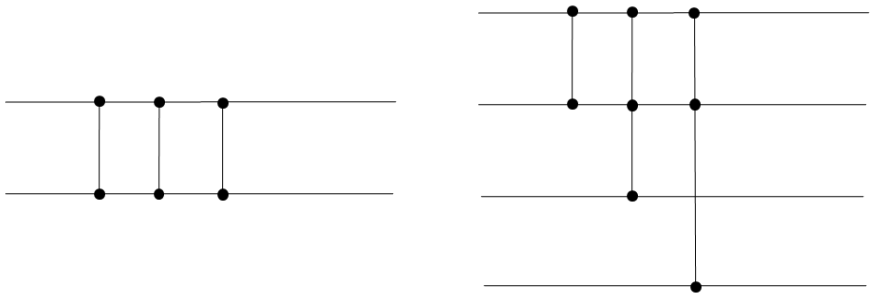

Fig. 5.1: Two examples of elements in $\mathcal{E}$ in generalized fMSA instances that may contain a chain of edges or even whole sub-alignments.

Of course the problem we want to solve on generalized fMSA instances is the following.

Definition 5.2
Let $(M, \mathcal{E}, \hat{w})$ be a generalized fMSA instance. We call the problem of finding a set $S \subset E$ with maximum weight $\hat{w}(S)$ the generalized fMSA-subgraph problem and S a generalized maximum fMSA-subgraph .

The definition above is in the same vein as the definition of the generalized maximum weight trace problem in [49], although it is slighty more general in the sense that the generalized maximum weight trace problem assesses the weight of a solution by the sum of weights of the chosen edges whereas the generalized fMSA-subgraph problem may have a different objective function $\hat{w}$.

The algorithmic framework is similar to the plane-sweep algorithm in the outer structure and also based on an iterative concept. During all iterations a set of possible candidate solutions $\mathcal{S}$ is computed that initially contains only the trivial candidate $S = \emptyset$ whereby the weight of the candidates are assessed by the generalized weight function $\hat{w}$. Essentially, the framework is parameterized by the following three sub-routines:

- $NEXT\_E()$: returns the next unseen edge set e from $\mathcal{E}$ to be consumed.
- $SPLIT\_CAND(S, E')$: has a solution candidate S and the current edge set $E' \in \mathcal{E}$ as parameter and returns a set of new possible solution candidates based on S and E' that may or may not contain E' or only parts of it.

- $CONS(\mathcal{S})$: returns a consolidated set of solution candidates $\mathcal{S}$.

Altogether we obtain the following algorithmic framework:

Algorithm 1 fMSA (M, $NEXT\_E(\cdot)$, $SPLIT\_CAND(\cdot,\cdot)$, $CONS(\cdot)$)

$\mathcal{S} \leftarrow \{\emptyset\}$
for all $E' \in \mathcal{E}$ **do**
  $SEEN(E') \leftarrow 0$
end for

while there is $E' \in \mathcal{E}$ with $SEEN(E') = 0$ **do**
  $E' \leftarrow NEXT\_E()$
  $\mathcal{S}' \leftarrow \emptyset$
  for all $S \in \mathcal{S}$ **do**
    $\mathcal{S}' \leftarrow \mathcal{S}' \cup SPLIT\_CAND(S, E')$
  end for
  $\mathcal{S} = CONS(\mathcal{S}')$
  $SEEN(E') \leftarrow 1$
end while

Let $\tilde{S} \in \mathcal{S}$ be a candidate with maximum weight $\hat{w}(S)$.
$RETURN \leftarrow \tilde{S}$

In all DIALIGN implementations [53, 52, 75, 74], fragments are represented by a chain of consecutive nodes in two sequences and therefore can easily be associated with elements of $\mathcal{E}$ whereby the weight is determined by the objective function w of DIALIGN. The set of solution candidates has size 1 throughout the whole algorithm in all DIALIGN variations and no *real* splits are considered at all due to the usage of a very trivial implementation of $SPLIT\_CAND$, i.e. either a fragment fits and an augmented candidate is returned or it raises a conflict and the candidate remains unchanged. Furthermore, the programs DIALIGN and DIALIGN-T each incorporate a greedy $NEXT_E()$ implementation while DIALIGN-TX employs a more sophisticated method of selecting the fragments (see chapter 6 and [74]) by combining greedy and progressive strategies. Analogously, the plane-sweep algorithm from chapter 4 obviously fits into this framework, too. However, in contrast to the DIALIGN methods, the plane-sweep algorithm considers *all* possibilities of inserting an edge e into a given candidate S when splitting the candidates while its 'Consolidate' step always keeps at least one candidate that is extensible to a maximum fMSA-subgraph.

5.2 The minimum removal problem

The DIALIGN programs, on the one hand make, use of a rather straightforward implementation whereas the plane-sweep algorithm, on the other hand, includes a very complex implementation $SPLIT\_CAND(S, e)$. However, in order to develop efficient heuristics, we are particularly interested in pragmatic approaches between those two extremes. Intuitively speaking, the question of what are the

lowest costs of inserting an edge e in a candidate solution S naturally arises, i.e. the problem of removing a minimum weighted subset of edges from S such that e can be inserted without violation the fMSA-subgraph definition. More formal, we can define this problem as follows:

Definition 5.3
Let $M = (G = (V, E), k, w, L = (l_1, l_2))$ be an fMSA instance, $G' = (V', E')$ an fMSA-subgraph of M and $e' = (u, v) \in E$. We are now interested in a subgraph $G'_{e'} = (V'_{e'}, E'_{e'})$ of G' such that $G'_{e'}$ together with e' yields a valid fMSA-subgraph of M. We call this the removal problem of (G', e') and we call $E' \backslash E'_{e'}$ a removal of (G', e'). The *minimum* removal problem is then given by finding a removal of (G', e') with minimum weight.

In the following, we show how the removal problem can be solved in polyonimal time by reducing it to a min-cut/max-flow instance $F_{e'} = (V_{e'}, E_{e'}, c_{e'})$ comprised of a set of nodes V, edges E between those nodes and a capacity function $c_{e'}$ that assigns a non-negative number $c_{e'}(e)$ to each edge $e \in E$. The construction of $F_{e'}$ is also illustrated in Figure 5.2. Initially, we remove all edges $e \in E'$ that do not lie on a common conflicting cycle with e'. We enumerate the layers from bottom to top (arbitrarily) such that the bottom and top layers are given by $l_1(u)$ and $l_1(v)$, respectively. For each layer, we introduce a leftmost and a rightmost node s_i and t_j and edges that connect consecutive nodes on each layer from s_i to t_j. On the bottom layer, the leftmost node is labeled s (the source) and the top layer the rightmost node is labeled t (the sink). Due to symmetry we can assume w.l.o.g. that the immediately next node on layer $l_1(u)$ of s is u itself and we remove the edges (s, u) and (v, t). Finally we assign flow capacities to the edges; the capacities of edges $e \in E$ are given by their weights $c(e) = w(e)$ and the capacities of the horizontal edges that connect to two nodes on the same layer are set to T where

$$T = k \cdot \left(1 + \sum_{e \in E'} w(e)\right)^2 .$$

Finally, we set the capacity of the leftmost and rightmost vertical edges connecting the nodes s, t, s_i, t_j to T^2. We know from section 4.4.1 that it requires time $O(n)$ to answer the question whether two edges are part of a common conflicting cycle. We thus need $O(n^2)$ time to build the max-flow instance $F_{e'} = (V_F, E_F, c)$.

Lemma 5.1
Let $\tilde{E} \subset E$ be a minimum cut of $F_{e'}$ with respect to s and t; then $\tilde{E} \cap E$ constitutes a minimum removal of (G', e').

Proof: Let $\tilde{E}$ be any removal of (G', e'); then by definition the insertion of e after removing $\tilde{E}$ yields an fMSA-subgraph, i.e. it allows all remaining edges of E, and especially e, to be drawn vertically. If we now transfer this drawing to the flow instance $(V_F, E_F \backslash \tilde{E})$ and additionally remove from every layer the unique horizontal edge that is crossed by e', we immediately receive a valid cut of $F_{e'}$. To conclude the proof of the lemma, we have now to show for a minimum cut $\tilde{E}$ of $F_{e'}$ that the set $\tilde{E} \cap E$ constitutes a valid removal of (G', e') compatible

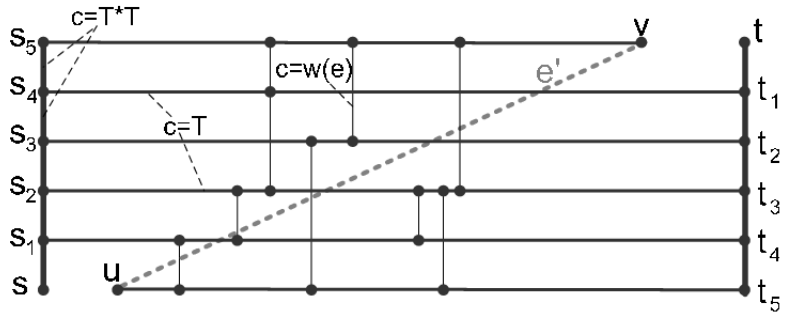

Fig. 5.2: The max-flow/min-cut problem induced by the fMSA removal problem.

with the canonical correspondence. Therefore we assume the opposite is true and thus can identify a set $D = (d_1, \ldots, d_{k-1})$ of edges in E' that together with e' form a conflicting cycle. We can assume that the edges in D are naturally ordered from bottom to top and enumerated exactly as such. Obviously, $\tilde{E}$ contains no edge connecting two s-nodes or two t-nodes of capacity T^2 since the removal of all other edges in E and exactly one horizontal edge of weight T in every layer, except on the top-most and bottom-most layer, would yield a cut of weight

$$(k-2) \cdot T + T < k \cdot T < T^2.$$

This observation also shows that on every layer except the bottom-most and top-most, *exactly* one horizontal edge of capacity T is contained in $\tilde{E}$. Let now x be the bottom-most layer such that the incoming node v_x (from the bottom) of an edge $d_x = (u_x, v_x) \in D$ can be reached by a path f from s that traverses only those vertical edges that interconnect s and s_i nodes. Since the edges (s, u) and (v, t) do not exist and due to our observation that exactly one horizontal edge of every layer is missing in E', the layer x cannot be the bottom layer and the top layer would meet the requirements for x if no other layer does. If we now add d_x to f, we can further extend f to a path that reaches t by traversing horizontally to the corresponding node t_i and from there to t, which is made possible by the choice of the bottom-most r and d_x (see also Figure 5.3). Finally, the existence of the path f is in contradiction to the fact of $\tilde{E}$ being a cut, which concludes the proof.

We are going to use the push-relabel maximum flow algorithm using Sleator's and Tarjan's dynamic tree data structure [70, 29] to solve the induced maximum flow problem. This algorithm has running time $O(VE \log(V^2/E))$, whereby V denotes the number of nodes and E the number of edges E. Let $n = |M|$ be the size of the fMSA instance. Then we get by our construction of the flow instance $|V| \leq O(n)$ and $|E| = O(n)$ and thus we conclude the following theorem.

Theorem 5.1
Let $M = (G = (V, E), k, w, L = (l_1, l_2))$ be an fMSA instance of size n, $G' = (V', E')$ an fMSA-subgraph of M and $e' = (u, v) \in E$. Then the above algorithm computes a minimum removal of (G', e') in time $O(n^2 \log(n))$.

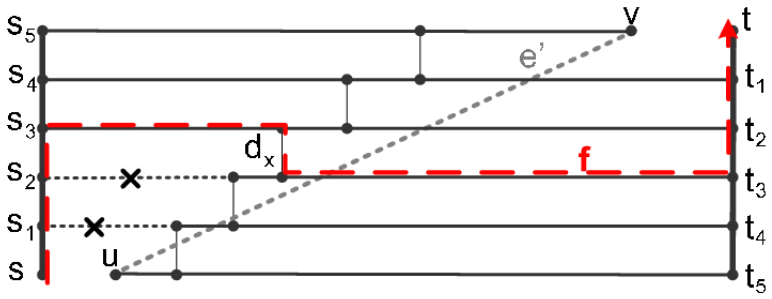

Fig. 5.3: Every conflicting path of e' has to be interrupted in an optimum min-cut since otherwise s and t can be connected by path f.

Coming back to the algorithmic framework, this efficient algorithm for finding minimum removals is suspected of being very useful in the 'Split' phase when developing new heuristics along the framework. For instance, let $S \in \mathcal{S}$ be a candidate solution and $e \in E$ the current edge to process; of course, the universal approach (as in the plane-sweep algorithm) would be to consider all possible removals of (S, e), as the plane-sweep algorithm does, whereas in the DIALIGN implementations no removal at all is taken into account. Between those extremes, the minimum removal R may give rise to a useful threshold for removals to look at in the 'Split' phase, e.g. removals with a weight sum that exceeds the weight sum of R by more than X percent are discarded.

5.3 Conclusion

The algorithmic framework for generalized instances of the fMSA-subgraph problem embraces various approaches that are based on assembling the alignment purely from the valuable pairwise similarities. In particular all DIALIGN implementations and the plane-sweep algorithm are prominent extremes. The simple DIALIGN implementations do not even seek subsets of a candidate solution in which an otherwise conflicting edge e (or fragment as a chain of edges) can be inserted, whereas the very complex but also time-consuming plane-sweep algorithm always considers *all* such alternatives. To close this chasm, we have presented a polynomial time algorithm for optimally inserting a given similarity edge into a given candidate solution as a tradeoff between those two extremes, which opens new possibilities for further developing sophisticated and efficient approximation algorithms using the algorithmic framework. Our framework is limited in the sense that there is no accounting for gaps between two different similarities in $\mathcal{E}$ and, at first glance, it therefore tends to be more appropriate for long and locally related sequence sets rather than for short and globally related ones. However, this restriction is only valid if the relevant gaps are not already implicitly covered by the given similarities in the sense that they are part of elements in the generalized similarity set $\mathcal{E}$ and penalized locally by the generalized weight function $\hat{w}$. In other words, the missing sensitivity for gaps in the outer routines of the algorithmic framework are conjectured to be partly compensated for by the careful choice of $\mathcal{E}$ and the definition of $\hat{w}$, which also asseses the rel-

evance of the candidate solutions. Additionally, the idea of a gapped alignment graph in [3] may be helpful in future in order to further reduce the bias towards locally related input sequences. We omitted in this chapter the discussion of the proper choice of the weight functions as well as the detailed handling of gaps and always assumed them to be part of our input. Obviously, there is a qualitative correlation between the strategy for selecting the input similarities, the weighting function and the assembly strategy and therefore we will focus in our future work on the development of efficient combined strategies for computing the relevant pairwise data along with a generalized weight function *and* the proper algorithmic parameterization of the framework for assembling them using the algorithmic framework. Depending on the specific biological domain one is interested in, such a stragegy would also be applicable for the development of alignment programs that are optimized towards special biological questions. Altogether, we conclude that the algorithmic framework gives us a rich toolset for future improvements to the segment-based approach for computing *general* and *(biological) domain-specific* multiple sequence alignments.

6. IMPROVEMENTS IN DIALIGN-TX

In chapter 3 we described DIALIGN-T, which is a reimplementation of the multiple-alignment program DIALIGN. Due to several algorithmic improvements, it produces significantly better alignments on locally and globally related sequence sets than previous versions of DIALIGN. However, like the original implementation of the program, DIALIGN-T uses a a straightforward greedy approach to assemble multiple alignments from local pairwise sequence similarities. Such greedy approaches may be vulnerable to spurious random similarities and can therefore lead to suboptimal results. In this chapter, we present DIALIGN-TX, a substantial improvement of DIALIGN-T that combines our previous greedy algorithm with a progressive alignment approach inspired by the algorithmic framework described in chapter 5. The results of this chapter have already been published in [74]

Our new heuristic produces significantly better alignments, especially on globally related sequences, without excessively increasing the CPU time and memory consumption. The new method is based on a guide tree; to detect possible spurious sequence similarities, it employs a vertex-cover approximation on a conflict graph. We performed benchmarking tests on a large set of nucleic acid and protein sequences. For protein benchmarks, we used the benchmark database BAliBASE 3 and an updated release of the database IRMBASE 2 for assessing the quality on globally and locally related sequences, respectively. For alignment of nucleic acid sequences, we used BRAliBase II for global alignment and a newly developed database of locally related sequences called *DIRMBASE 1*. IRMBASE 2 and DIRMBASE 1 are constructed by implanting highly conserved motifs at random positions in long unalignable sequences.

On BAliBASE3, our new program DIALIGN-TX performs significantly better than the previous program DIALIGN-T and outperforms the popular global aligner CLUSTAL W, though it is still outperformed by programs that focus on global alignment like MAFFT, MUSCLE and T-COFFEE. On the locally related test sets in IRMBASE 2 and DIRMBASE 1, our method outperforms all other programs while MAFFT E-INSi is the only method that comes close to the performance of DIALIGN-TX.

6.1 Assembling alignments from fragments

The major improvement in DIALIGN-TX, compared to DIALIGN-T, lies in a new strategy for assembling the fragments using progressive and greedy ap-

proaches, whereas DIALIGN-T is only based on a greedy approach (see chapter 3).

Formally, we consider the following optimization problem: we are given a set $S = \{s_1, \ldots, s_k\}$ of input sequences where l_i is the length of sequence s_i. A fragment f is a pair of two equal-length segments from two different input sequences. Thus, a fragment represents a local pairwise gap-free alignment of these two sequences. Each possible fragment f is assigned a *weight score* $w(f)$ which, in our approach, depends on the probability $P(f)$ of *random occurrence* of such a fragment. More precisely, if f is a local alignment of sequences s_i and s_j, then $P(f)$ is the probability of finding a fragment of the same length as f with at least the same sum of matches or similarity values for amino acids in *random* sequences of length l_i and l_j, respectively. For protein alignment, a standard substitution matrix is used and for DNA alignment a simple match-mismatch matrix is taken.

Let F be the set of all possible fragments. The optimization problem is then to find a *consistent* set $A \subset F$ of fragments with maximum total weight, i.e. a consistent set A maximizing

$$W(A) := \sum_{f \in A} w(f).$$

A set of fragments is called *consistent* if all fragments can be included into one single alignment; see [58]. Fragments in A are allowed to overlap if different pairs of sequences are involved. That is, if two fragments $f_1, f_2 \in A$ involve sequence pairs s_i, s_j and s_j, s_k, respectively, then f_1 and f_2 are allowed to overlap in sequence s_j. If two fragments involve the same pair of sequences, no overlap is allowed. It can be shown that the problem of finding an optimal consistent set A of fragments is NP-complete (Constructing multiple sequence alignments from pairwise data, Subramanian *et al.*, in preparation). Therefore, we are motivated in finding intelligent approximations that deliver a good tradeoff between alignment quality and CPU time.

To decrease the computational complexity of this problem, we restrict ourselves to a reduced subset $F' \subset F$ and we will first search for a consistent subset $A \subset F'$ with maximum total score. As in previous versions of DIALIGN, we use pairwise optimal alignments as a filter. In other words, the set F' is defined as the set of all fragments contained in any of the optimal *pairwise* alignments of the sequences in our input data set. Here, we also restrict the length of fragments using some suitable constant.

For multiple alignment, previous versions of DIALIGN used the above outlined *greedy* approach. We call this approach a *direct* greedy approach, as opposed to the *progressive* greedy approach that we introduce in this chapter. A modification of this *direct greedy* approach was also used in our reimplementation DIALIGN-T. Here, we considered not only the weight scores of individual fragments (or their *overlap* weights [53]) but also took into account the overall degree of similarity between the two sequences involved in the fragment. The rationale behind this approach is that a fragment from a sequence pair with high overall similarity is less likely to be a random artefact than a fragment from an otherwise non-related sequence pair which is undermined by the positive effect of the weight score factors introduced in DIALIGN-T; see chapter 3.

6.1.1 Combining segment-based greedy and progressive alignments

To overcome the difficulties of a *direct* greedy algorithm for multiple alignment, we combined greedy features with a *progressive* alignment approach [26, 76, 14, 40]. Roughly, the new method we developed first computes a *guide tree* for the set of input sequences based on their pairwise similarity scores. The sequences are then aligned in the order defined by the guide tree. We divide the set of fragments contained in the respective optimal pairwise alignments into two subsets F_0 and F_1 where F_0 consists of all fragments with weight scores *below* the average fragment score in all pairwise alignments, and F_1 consists of the fragments with a weight above or equal to the average weight. In a first step, the set F_1 is used to calculate an initial multiple alignment A_1 in a *progressive* manner. The low-scoring fragments from set F_0 are added later to A_1 in a *direct* greedy way, provided they are consistent with A_1. In addition, we construct an alternative multiple alignment A_0 using the *direct* greedy approach implemented in previous versions of DIALIGN and DIALIGN-T. The program finally returns either A_0 or A_1, depending on which one of these two alignments has the highest score.

To construct a guide tree for the progressive alignment algorithm, we use straightforward hierarchical clustering. Here, we use a weighted combination of complete-linkage and average-linkage clustering based on pairwise similarity values $R(p, q)$ for pairs of cluster (C_p, C_q). Initially, each cluster C_i consists of one sequence s_i only. The similarity $R(i, j)$ between clusters C_i and C_j (or leaves i and j in our tree) is defined to be the score of the optimal pairwise alignment of s_i and s_j according to our objective function, i.e. the sum of the weights of the fragments in an optimal chain of fragments for these two sequences. In every step, we merge the two sequence clusters C_i and C_j with the maximum similarity value $R(i, j)$ into a new cluster. Whenever a new cluster C_p is created by merging clusters q and r (or a node p in the tree is created with children q and r), we define the similarity between p and all other remaining clusters m to be

$$R(m,p) := 0.1 \cdot \frac{1}{2}\left(R(m,p) + R(m,q)\right) + 0.9 \cdot \max\left(R(m,p), R(m,q)\right).$$

The choice of this function was inspired by MAFFT [43, 42]; it also worked very well in our situation on globally and locally related sequences after experiments on BAliBASE 3, BRAliBase II, IRMBASE 2 and DIRMBASE 1.

6.1.2 Merging two sub-alignments

The final multiple alignment of our input sequence set S is constructed bottom-up along the guide tree. Thus, the crucial step is to combine two sub-alignments represented by nodes q and r in our tree whenever a new node p is created. In the traditional *progressive* alignment approach, this is done by calculating a pairwise alignment of *profiles*, but this procedure cannot be directly adapted to our segment-based approach. Let A_q and A_r be the existing subalignments of the sequences in clusters C_q and C_r, respectively, at the time where these clusters are merged to a new cluster C_p. Let $F_{q,r}$ be the set of all fragments $f \in F$ connecting one sequence from cluster C_q with another sequence from

cluster C_r. Now, our main goal is to find a subset $F_p \subset F_{q,r}$ with maximum total weight score that is consistent with the existing alignments A_q and A_r. In other words, we are looking for a subset $F_p \subset F_{q,r}$ with maximum total weight such that

$$A_p = A_q \cup A_r \cup F_p$$

describes a valid multiple sequence alignment of the sequence set represented by node p.

It is easy to see at this time that before clusters A_q and A_r are merged, *every single* fragment $f \in F_{q,r}$ is consistent with the existing (partial) alignments A_q and A_r and therefore consistent with the set of all fragments accepted so far. Only *groups* of at least two fragments from $F_{q,r}$ can lead to inconsistencies with the previously accepted fragments. Thus, there are different subtypes of consistency conflicts in $F_{q,r}$ that may arise when A_q and A_r are fixed. There are pairs, triples or, in general, l-tuples of fragments of $F_{q,r}$ that give rise to a conflict in the sense that the conflict can be resolved by removing exactly one fragment of such a conflicting l-tuple. Statistically, pairs of conflicting fragments are the most frequent type of conflict, so we will take care of them more intelligently rather than using only a greedy method. Since, in our approach, the length of fragments is limited, we can easily determine in constant time for any pair of fragments (f_1, f_2) if the set

$$A_q \cup A_r \cup \{f_1, f_2\}$$

is consistent, i.e. if it forms a valid alignment, or if there is a pairwise conflict between f_1 and f_2. Here, the data structures described in [1] are used. With unbounded fragment length, the consistency check for the new fragments (f_1, f_2) would take $O(|f_1| \times |f_2|)$ time where $|f|$ is the length of a fragment f.

This gives rise to a conflict graph $G_{q,r}$ that has a weighted node n_f for every fragment $f \in F_{q,r}$. The weight $w(n_f)$ of node n_f is defined to be the weight score $w(f)$ of f, and for any two fragments f_1, f_2 there exists an edge connecting n_{f_1} and n_{f_2} iff there is a *pairwise conflict* between f_1 and f_2, i.e. if the set $A_q \cup A_r \cup \{f_1, f_2\}$ is *inconsistent*. We are now interested in finding a good subset of $F_{q,r}$ that does not contain any pairwise conflicts in the above sense. The optimum solution would be obtained by removing a minimally weighted vertex cover from $G_{q,r}$. Since the weighted vertex cover problem is NP-complete we apply the 2-approximation given by Clarkson [11]. This algorithm roughly works as follows: in order to obtain the vertex cover C, the algorithm iteratively adds the node v with the maximum value

$$\frac{degree(v)}{w(v)}$$

to C. For any edge (v, u) that connects a node u with v, the weight $w(u)$ is updated to

$$w(u) := w(u) - \frac{degree(v)}{w(v)}$$

and the edge (u, v) is deleted. This iteration is followed as long as there are edges left.

Note that it is *not* sufficient to remove the vertex cover C from $F_{q,r}$ to obtain a valid alignment since, in the construction of C, only inconsistent *pairs* of fragments were considered. We therefore first remove C from $F_{q,r}$ and we subsequently remove further inconsistent fragments from $F_{q,r}$ using our *direct* greedy alignment as described in chapter 3. A consequence of this further reduction of the set $F_{q,r}$ is that fragments that were previously removed because of pairwise inconsistencies may became consistent again. A node n_{f_1} may have been included into the set C and therefore removed from the alignment because the corresponding fragment f_1 is part of an inconsistent fragment pair (f_1, f_2). However, after subtracting the set C from $F_{q,r}$, the algorithm may detect that fragment f_2 is part of a larger inconsistent group, and f_2 is removed as well. In this case, it may be possible to include f_1 again into the alignment. Therefore, our algorithm reconsiders, in the final step, the set C to see if some of the previously excluded fragments can now be reincluded into the alignment. This is again done using our *direct* greedy method.

6.1.3 The overall algorithm

In the previous section, we discussed all ingredients that are necessary to give a high-level description of our algorithm to compute a multiple sequence alignment. For clarity, we omit algorithmical details and data structures such as the *consistency frontiers* that are used to check for consistency because these features have been already described in chapter 3.
We use a subroutine $PAIRWISE\_ALIGNMENT(s_i, s_j, A)$ that takes two sequences s_i and s_j and (optionally) an existing consistent set of fragments A as input and calculates an optimal alignment of s_i and s_j under the secondary constraint that this alignment is consistent with A and that only those positions in the sequences are aligned that are not yet aligned by a fragment from A. Note that in DIALIGN, an alignment is defined as an equivalence relation on the set of all sequence positions, so a consistent set of fragments corresponds to an alignment. Therefore, we do not formally distinguish between alignments and sets of fragments.

Next, a subroutine $GREEDY\_ALIGNMENT(A, F')$ takes an alignment A and a set of fragments F' as arguments and returns a new alignment $A' \supset A$ by adding fragments from the set F' in a *directly* greedy fashion. For details on these subroutines see also [75]. Furthermore, we use a subroutine $BUILD\_UPGMA(F')$ that takes a set F' of fragments as arguments and returns a tree and a subroutine $MERGE(p, F')$ that takes the parent node p and the set of fragments F' as arguments and returns an alignment of the set of sequences represented by node p. Those two subroutines have been described in the previous two subsections. A pseudo-code description of the complete algorithm for multiple alignment is given in Algorithm 2. The algorithm calculates a first alignment A_0 using our novel *progressive* approach and a second alignment A_1 with the greedy method previously used in DIALIGN. Finally, the alignment with the higher numerical score is returned. For the progressive method, *fragments*, i.e. local gap-free pairwise alignments from the respective optimal pairwise alignments are considered. Fragments with a weight score above the average fragment score are processed, first following a *guide tree* as described in the main text. Lower-scoring fragments are added later, provided they are

consistent with the previously included high-scoring fragments. Note that the output of the sub-routine *PAIRWISE_ALIGNMENT* is a chain of fragments. This is equivalent to a pairwise alignment in the sense of DIALIGN.

As in the original version of DIALIGN [53], the process of pairwise alignment and consistency filtering is carried out iteratively. Once a valid alignment A has been constructed by removing inconsistent fragments from the set F' of the fragments that are part of the respective optimal pairwise alignments, this procedure is repeated until no new fragments can be found. In the second and subsequent iteration steps, only those parts of the sequences that are not yet aligned are considered and optimal pairwise alignments are calculated under the consistency constraints imposed by the existing alignment A.

6.2 Further program features

Beside the improvements to the optimization algorithm described above, we incorporated new features into DIALIGN-TX that were already part of the original implementation of DIALIGN. DIALIGN-TX now supports *anchor points* the same way DIALIGN 2.2 does [56, 58]. Anchor points can be used for various purposes, e.g. to speed up alignment of large genomic sequences [10, 8], or to incorporate information about locally conserved motifs. This can be done, for example, using the *N-local-decoding* approach [12, 18] or other methods for motif finding.

DIALIGN-TX also now comes with an option to specify a threshold parameter T in order to exclude low-scoring fragments from the alignment. Following an approach proposed in [57], the alignment procedure can be iterated, starting with a high value of T and with lower values in subsequent iteration steps. By default, in the first iteration step of our algorithm, we use a value of $T = -\log_2(0.5)$ for the pairwise alignment phase, while in all subsequent iteration steps, a value of $T = 0$ is used. With a user-specified threshold of $T = 2$ for the first iteration step, the threshold value remains $-log_2(0.5)$ in all subsequent steps, and with a chosen threshold value of $T = 1$, the value for the subsequent iteration steps is set to $-log_2(0.75)$.

An optimal pairwise alignment in the sense of our segment-based approach is a chain of fragments with maximum total weight score. Calculating such an optimal alignment takes $O(l^3)$ time where l denotes the (maximum) length of the two sequences since all possible fragments are to be considered. If the length of fragments is bounded by a constant L, the complexity is reduced to $O(l^2 \times L)$. In practice, however, it is not meaningful to consider all possible fragments. Our algorithm processes fragments starting at a pair of positions i and j with increasing fragment length. To reduce the number of fragments considered, our algorithm stops processing longer fragments starting at i and j if the previously visited short fragments starting at the same positions have low scores. More precisely, we consider the average substitution score of aligned amino acids or the average number of matches for DNA or RNA alignment, respectively, to decide if further fragments starting at i and j are considered.

To reduce the run time for pairwise alignments, we implemented an option called *fast mode*. This option uses a lower threshold value for the average substitu-

Algorithm 2 DIALIGN-TX $(s_1, \ldots, s_k)$

$F \leftarrow \emptyset$
for all s_i, s_j such that $i < j$ **do**
  $F \leftarrow F \cup PAIRWISE\_ALIGNMENT(s_i, s_j, \emptyset)$
end for

/* initial computation of A_1: original DIALIGN alignment */
$A_1 \leftarrow \emptyset$
$A_1 \leftarrow GREEDY\_ALIGNMENT(A_1, F)$

/* initial computation of A_0: "progressive DIALIGN" alignment */
$a = AVERAGE(w(f)|f \in F)$
$F_0 = \{f \in F | w(f) < a\}$
$F_1 = \{f \in F | w(f) \geq a\}$
$T = BUILD\_UPGMA(F)$

while there is an unprocessed non-leaf node in T **do**
  Let p be an unprocessed non-leaf node such that the child-nodes are either marked as processed or are leaves.
  $A'(p) \leftarrow MERGE(p, F_1)$
  $PROCESSED(p) \leftarrow TRUE$
end while

$A_0 \leftarrow A'(ROOT(T))$
$A_0 \leftarrow GREEDY\_ALIGNMENT(A_0, F_0)$

/* adding further fragments to A_1 */
while additional fragments can be found **do**
  $F \leftarrow \emptyset$
  for all s_i, s_j such that $i < j$ **do**
    $F \leftarrow F \cup PAIRWISE\_ALIGNMENT(s_i, s_j, A_1)$
  end for
  $A_1 \leftarrow GREEDY\_ALIGNMENT(A_1, F)$
end while

/* adding further fragments to A_0 */
while additional fragments can be found **do**
  $F \leftarrow \emptyset$
  for all s_i, s_j such that $i < j$ **do**
    $F \leftarrow F \cup PAIRWISE\_ALIGNMENT(s_i, s_j, A_0)$
  end for
  $A_0 \leftarrow GREEDY\_ALIGNMENT(A_0, F)$
end while

if $W(A_0) > W(A_1)$ **then**
  $RETURN \leftarrow A_0$
else
  $RETURN \leftarrow A_1$
end if

tion scores or number of matches. By default, during the pairwise alignment phase, fragments under consideration are extended until their average substitution score is at least 4 for amino acids (note that our BLOSUM62 matrix has 0 for the lowest score possible) and 0.25 for nucleotides. With the *fast mode* option, this threshold is increased by 0.25, which has the effect that the extension of fragments during the pairwise alignment phase is interrupted far more often than by default. This option, however, reduces the sensitivity of the program. We observed speed-ups up to factor 10 on various benchmark data when using this option while the alignment quality was still reasonably high, in the sense that the average sum-of-pair score and average column score on our benchmarks deteroriated by around $5\% - 10\%$ only. We recommend using this option for large input data containing sequences that are not too distantly related. Hence, this option is not advisable for strictly locally related sequences, where we observed a reduction of the alignment quality almost down to a score of zero. However, in the latter case, this option is not necessary since the original similarity score thresholds of 4 for amino acids and 0.25 for nucleotides are effective enough to prevent DIALIGN-TX from unnecessarily looking at too many spurious fragments.

6.3 Benchmark results

In order to evaluate the improvements of the new heuristics, we had several benchmarks on various reference sets and compared DIALIGN-TX with its predecessor DIALIGN-T 0.2.2 [75], DIALIGN 2.2 [51], CLUSTAL W2 [77], MUSCLE 3.7 [25], T-COFFEE 5.56 [60] POA V2 [35, 48], PROBCONS 1.12 [20] & PROBCONSRNA 1.10, MAFFT 6.240 L-INSi and E-INSi [43, 42]. We performed benchmarks for DNA as well as for protein alignment. As globally related benchmark sets we used BRAliBase II [27, 88] for RNA and BAliBASE 3 [78] for protein sequences.

The benchmarks on locally related sequence sets were run on IRMBASE 2 for proteins and DIRMBASE 1 for DNA sequences, which have been constructed in a very similar way as IRMBASE 1 [75] by implanting highly conserved motifs generated by ROSE [73] in long random sequences. IRMBASE 2 and DIRMBASE 1 both consist of four reference sets ref1, ref2, ref3 and ref4 with one, two, three and four (respectively) randomly implanted ROSE motifs. The major difference compared to the old IRMBASE 1 lies in the fact that in $1/s$ cases the occurrence of a motif in a sequence has been omitted randomly, whereby s is the number of sequences in the sequence family. The benchmark results for IRMBASE 2 and DIRMBASE 1 now tell us how the alignment programs perform in cases when it is unknown whether every motif occurs in every sequence, thus providing a more realistic basis for assessing the alignment quality on locally related sequences compared to the situation in the old IRMBASE 1 where every motif *always* occurred in *every* sequence.

Each reference set in IRMBASE 2 and DIRMBASE 1 consists of 48 sequence families, 24 of which contain ROSE motifs of length 30 while the remaining families contain motifs of length 60. Sixteen sequence families in each of the reference sets consist of 4 sequences each, another 16 families consist of 8 sequences while the remaining 16 families consist of 16 sequences. In ref1, random

sequences of length 400 are added to the conserved ROSE motif while for ref2 and ref3, random seqences of length 500 are added. In ref4 random sequences of length 600 are added.

For both BAliBASE and IRMBASE, we used two different criteria to evaluate multi-alignment software tools. We used the *sum-of-pair score* (SPS) where the percentage of correctly aligned *pairs* of residues is taken as a quality measure for alignments. In addition, we used the *column score* (CS) where the percentage of correct *columns* in an alignment is the criterion for alignment quality. Both scoring schemes were restricted to *core blocks* within the reference sequences where the 'true' alignment is known. For IRMBASE 2 and DIRMBASE 1, the core blocks are defined as the conserved ROSE motifs. To compare the output of different programs to the respective benchmark alignments, we used C. Notredame's program `aln_compare` [60].

6.3.1 Results on locally related sequence families

The quality results of our benchmarks of DIALIGN-TX and various alignment programs on the local aligment databases can be found in Tables 6.1 and 6.2 for the local protein database IRMBASE 2 and in Tables 6.3 and 6.4 for the local DNA database DIRMBASE 1. The average CPU times of the tested methods are listed in Table 6.5. When looking at the results, DIALIGN-TX clearly outperforms all other methods on sum-of-pairs score (SPS) and column score (CS) with the only exception that MAFFT E-INSi outperforms DIALIGN-TX on the SPS on IRMBASE 2 whilst in turn DIALIGN-TX is around 3.5 times faster and significantly outperforms MAFFT-EINSi on the CS. The superiority of DIALIGN-TX compared to DIALIGN-T 0.2.2 is not statistically significant on IRMBASE 2, however, it is on DIRMBASE 1 which is due to a very low sensitivity threshold parameter for the DNA case set by default in DIALIGN-T 0.2.2, which allowed fragments solely comprised of matches. In *all* other comparisons DIALIGN-TX is significantly superior to the other programs with respect to the Wilcoxon Matched Pairs Signed Rank Test [87]. DIALIGN 2.2, DIALIGN-T 0.2.2 (only for protein), MAFFT L-INSi and MAFFT E-INSi were the only other methods that produced reasonable results.

On IRMBASE 2 our new program DIALIGN-TX is around 1.64 times slower compared to DIALIGN-T, however, it is still faster than DIALIGN 2.2. On DIRMBASE 1 we observed that DIALIGN-TX is 4.26 times slower than DIALIGN-T (which is due to the reduced sensitivity in DIALIGN-T 0.2.2) and we also see that DIALIGN-TX is around 2.04 times slower than DIALIGN 2.2. Although IRMBASE 2 and DIRMBASE 1 are constructed in a similar way, we see that T-COFFEE and PROBCONS behave quite well on the protein alignments whereas they perform very poorly in the DNA case, while the other methods ranked about equally in the protein and DNA cases. Overall, we conclude from our benchmarks that DIALIGN-TX is the dominant program on locally related sequence protein and DNA families that consist of closely related motifs embedded in long unalignable sequences.

Tab. 6.1: Sum-of-pairs scores of various alignment programs on the benchmark database IRMBASE 2

Method (Protein)	REF1	REF2	REF3	REF4	Total
DIALIGN-TX	89.42	**94.90**	93.75	93.64	92.93
DIALIGN-T 0.2.2	89.67^{0}	94.19^{0}	$\mathbf{93.93}^{0}$	93.12^{0}	92.73^{0}
DIALIGN 2.2	90.43^{0}	93.40^{-}	91.78^{--}	92.98^{-}	92.15^{--}
CLUSTAL W2	07.13^{--}	10.63^{--}	19.87^{--}	26.17^{--}	15.95^{--}
T-COFFEE 5.56	72.67^{--}	77.80^{--}	83.03^{--}	83.48^{-}	79.24^{--}
POA V2	87.56^{-}	49.57^{--}	41.90^{--}	37.56^{--}	54.15^{--}
MAFFT 6.240 L-INSi	82.78^{0}	84.29^{-}	84.15^{--}	82.42^{--}	84.41^{--}
MAFFT 6.240 E-INSi	$\mathbf{90.53}^{0}$	94.37^{0}	93.11^{0}	$\mathbf{94.79}^{+}$	$\mathbf{93.20}^{+}$
MUSCLE 3.7	32.67^{--}	34.82^{--}	54.19^{--}	57.84^{--}	44.88^{--}
PROBCONS 1.12	78.78^{--}	86.82^{--}	87.29^{-}	87.69^{--}	85.15^{--}

Average sum-of-pair scores (SPS) of the benchmarked programs on the core blocks (given by the implanted conserved motifs) of IRMBASE 2. *Minus* symbols denote statistically significant *inferiority* of the respective method compared with DIALIGN-TX, while *plus* symbols denote statistically significant superiority of the method. ^{0} denotes non-significant superiority or inferiority of DIALIGN-TX. Single plus or minus symbols denote significance according to the Wilcoxon Matched Pairs Signed Rank Test with $p \leq 0.05$ and double symbols denote significance with $p \leq 0.001$.

Tab. 6.2: Column scores of different programs on IRMBASE 2

Method (Protein)	REF1	REF2	REF3	REF4	Total
DIALIGN-TX	64.17	**77.36**	70.30	**72.23**	**71.02**
DIALIGN-T 0.2.2	67.04^{0}	75.81^{0}	70.40^{0}	$\mathbf{70.44}^{0}$	70.93^{0}
DIALIGN 2.2	$\mathbf{68.52}^{0}$	73.32^{-}	65.34^{-}	69.50^{-}	69.17^{--}
CLUSTAL W2	00.00^{--}	00.00^{--}	00.11^{--}	02.86^{--}	00.74^{--}
T-COFFEE 5.56	34.84^{--}	40.87^{--}	43.62^{--}	49.56^{--}	42.22^{--}
POA V2	50.99^{-}	16.95^{--}	11.79^{--}	10.18^{--}	22.47^{--}
MAFFT 6.240 L-INSi	37.81^{--}	39.54^{--}	32.79^{--}	38.75^{--}	32.22^{--}
MAFFT 6.240 E-INSi	45.70^{-}	52.37^{--}	43.11^{--}	54.82^{--}	49.00^{--}
MUSCLE 3.7	04.65^{--}	06.87^{--}	14.80^{--}	19.65^{--}	11.49^{--}
PROBCONS 1.12	36.77^{--}	43.47^{--}	41.89^{--}	43.56^{--}	41.42^{--}

Average column scores (CS) of the benchmarked programs on the core blocks of IRMBASE 2. The symbols are analogous to Table 6.1.

Tab. 6.3: Sum-of-pairs scores on DIRMBASE 1

Method (DNA)	REF1	REF2	REF3	REF4	Total
DIALIGN-TX	**94.38**	**92.85**	**95.44**	**95.70**	**94.59**
DIALIGN-T 0.2.2	64.00^{--}	61.22^{--}	64.96^{--}	65.24^{--}	63.85^{--}
DIALIGN 2.2	92.61^{-}	91.10^{-}	94.62^{-}	94.13^{-}	93.12^{--}
CLUSTAL W2	06.79^{--}	08.27^{--}	18.51^{--}	29.09^{--}	15.66^{--}
T-COFFEE 5.56	14.71^{--}	18.88^{--}	32.08^{--}	43.39^{--}	27.62^{--}
POA V2	32.03^{--}	27.40^{--}	28.78^{--}	32.18^{--}	30.10^{--}
MAFFT 6.240 L-INSi	52.40^{--}	48.81^{--}	49.77^{--}	57.47^{--}	52.36^{--}
MAFFT 6.240 E-INSi	92.42^{0}	84.15^{--}	87.91^{-}	89.36^{-}	88.46^{--}
MUSCLE 3.7	48.17^{--}	54.40^{--}	56.57^{--}	60.24^{--}	56.84^{--}
PROBCONSRNA 1.10	13.00^{--}	12.94^{--}	20.28^{--}	32.56^{--}	19.69^{--}

Average sum-of-pair scores (SPS) of the benchmarked programs on the core blocks of DIRMBASE 1 . The symbols are analogous to Table 6.1.

Tab. 6.4: Column scores on DIRMBASE 1.

Method (DNA)	REF1	REF2	REF3	REF4	Total
DIALIGN-TX	**74.39**	**69.03**	**71.57**	**75.11**	**72.52**
DIALIGN-T 0.2.2	29.60^{--}	28.63^{--}	35.51^{--}	35.85^{--}	32.40^{--}
DIALIGN 2.2	69.95^{0}	68.19^{0}	71.25^{0}	72.48^{0}	70.47^{-}
CLUSTAL W2	00.00^{--}	00.00^{--}	02.19^{--}	04.99^{--}	01.80^{--}
T-COFFEE 5.56	00.00^{--}	00.18^{--}	04.01^{--}	08.44^{--}	03.16^{--}
POA V2	05.63^{--}	07.32^{--}	04.12^{--}	06.81^{--}	05.97^{--}
MAFFT 6.240 L-INSi	21.45^{--}	11.93^{--}	16.02^{--}	22.30^{--}	17.93^{--}
MAFFT 6.240 E-INSi	40.28^{--}	41.99^{--}	45.77^{--}	51.01^{--}	44.76^{--}
MUSCLE 3.7	14.18^{--}	16.18^{--}	19.62^{--}	30.43^{--}	20.10^{--}
PROBCONSRNA 1.10	00.73^{--}	00.05^{--}	01.34^{--}	04.31^{--}	01.61^{--}

Average column scores (CS) of the benchmarked programs on the core blocks of DIRMBASE 1. The symbols are analogous to Table 6.1.

Tab. 6.5: Program run time on IRMBASE 2 and DIRMBASE 1

Method	Average runtime on IRMBASE 2	Average runtime on DIRMBASE 1
DIALIGN-TX 1.0	4.47	9.84
DIALIGN-T 0.2.2	2.73	2.31
DIALIGN 2.2	4.98	4.82
CLUSTAL W2	1.86	1.36
T-COFFEE 5.56	26.41	365.88
POA V2	1.81	1.20
MAFFT 6.240 L-INSi	8.47	5.33
MAFFT 6.240 E-INSi	15.35	8.39
MUSCLE 3.7	6.34	4.87
PROBCONS(RNA) 1.12(1.10)	28.27	18.54

Average running time (in seconds) per multiple alignment for sequence families on IRMBASE 2 and DIRMBASE 1. Program runs were performed on a Linux workstation with a 3.2 GHz Pentium 4 processor and 2 GB RAM.

Tab. 6.6: Sum-of-pairs scores on BAliBASE 3

Method (Protein)	RV11	RV12	RV20	RV30	RV40	RV50	Total
DIALIGN-TX	51.52	89.18	87.87	76.18	83.65	82.28	78.83
DIALIGN-T 0.2.2	49.30^{-}	88.76^{0}	86.29^{0}	74.66^{0}	81.95^{-}	80.14^{-}	77.31^{--}
DIALIGN 2.2	50.73^{0}	86.66^{-}	86.91^{0}	74.05^{0}	83.31^{0}	80.69^{0}	77.52^{--}
CLUSTAL W2	50.06^{0}	86.43^{0}	85.16^{0}	72.50^{-}	78.93^{0}	74.24^{-}	75.36^{--}
T-COFFEE 5.56	58.22^{++}	92.27^{++}	90.92^{++}	79.09^{+}	86.03^{+}	86.09^{+}	82.41^{++}
POA V2	37.96^{--}	83.19^{--}	85.28^{-}	71.93^{-}	78.22^{--}	71.49^{--}	72.17^{--}
MAFFT 6 L-INSi	$\mathbf{67.11}^{++}$	93.63^{++}	$\mathbf{92.67}^{++}$	85.55^{++}	$\mathbf{91.97}^{++}$	$\mathbf{90.00}^{++}$	$\mathbf{87.07}^{++}$
MAFFT 6 E-INSi	66.00^{++}	93.61^{++}	92.64^{++}	$\mathbf{86.12}^{++}$	91.46^{++}	89.91^{++}	86.83^{++}
MUSCLE 3.7	57.90^{+}	91.67^{++}	89.17^{+}	80.60^{+}	87.26^{+}	83.39^{0}	82.19^{++}
PROBCONS 1.12	66.99^{++}	$\mathbf{94.12}^{++}$	91.68^{++}	84.61^{++}	90.24^{++}	89.28^{++}	86.40^{++}

Average sum-of-pair scores (SPS) of the benchmarked programs on the core blocks of BAliBASE 3. The symbols are analogous to Table 6.1.

Tab. 6.7: Column scores on BAliBASE 3

Method (Protein)	RV11	RV12	RV20	RV30	RV40	RV50	Total
DIALIGN-TX 1.0	26.53	75.23	30.49	38.53	44.82	46.56	44.34
DIALIGN-T 0.2.2	25.32^{0}	72.55^{0}	29.20^{0}	34.90^{-}	45.23^{0}	44.25^{0}	42.76^{-}
DIALIGN 2.2	26.50^{0}	69.55^{-}	29.22^{0}	31.23^{-}	44.12^{0}	42.50^{-}	41.49^{--}
CLUSTAL W2	22.74^{0}	71.59^{0}	21.98^{0}	27.23^{-}	39.55^{0}	30.75^{-}	37.35^{--}
T-COFFEE 5.56	31.34^{0}	81.18^{++}	37.81^{+}	36.57^{0}	48.20^{0}	50.63^{0}	48.54^{++}
POA V2	15.26^{--}	63.84^{--}	23.34^{-}	28.23^{-}	33.67^{--}	27.00^{--}	33.37^{--}
MAFFT 6 L-INSi	$\mathbf{44.61}^{++}$	83.75^{++}	$\mathbf{45.27}^{++}$	56.93^{++}	$\mathbf{59.69}^{++}$	56.19^{+}	$\mathbf{58.57}^{++}$
MAFFT 6 E-INSi	43.71^{++}	83.43^{++}	44.63^{++}	$\mathbf{58.80}^{++}$	58.33^{++}	$\mathbf{58.94}^{++}$	58.37^{++}
MUSCLE 3.7	33.03^{+}	80.46^{++}	35.22^{0}	38.77^{0}	45.96^{0}	44.94^{0}	47.58^{++}
PROBCONS 1.12	41.68^{++}	$\mathbf{85.52}^{++}$	40.49^{++}	54.37^{++}	52.90^{++}	56.50^{++}	55.66^{++}

Average column scores (CS) of the benchmarked programs on the core blocks of BAliBASE 3. The symbols are analogous to Table 6.1.

Tab. 6.8: Sum-of-pairs scores on BRAliBase II

Method (DNA)	G2In	rRNA	SRP	tRNA	U5	Total
DIALIGN-TX 1.0	72.08	91.69	82.92	78.53	77.80	80.42
DIALIGN-T 0.2.2	54.68^{--}	69.13^{--}	60.81^{--}	64.44^{--}	67.87^{--}	63.53^{--}
DIALIGN 2.2	71.72^{0}	89.89^{--}	81.47^{--}	78.57^{0}	76.16^{--}	79.37^{--}
CLUSTAL W2	72.68^{0}	93.25^{+}	87.40^{++}	86.96^{++}	79.56^{+}	83.80^{++}
T-COFFEE 5.56	73.79^{0}	90.94^{+}	83.90^{0}	81.65^{0}	79.13^{+}	81.73^{+}
POA V2	67.22^{--}	88.92^{--}	85.47^{++}	76.91^{-}	77.28^{0}	79.02^{--}
MAFFT 6.240 L-INSi	78.93^{++}	93.85^{+}	87.46^{++}	91.79^{++}	82.80^{++}	86.84^{++}
MAFFT 6.240 E-INSi	77.39^{++}	93.80^{+}	87.24^{++}	90.60^{++}	80.46^{++}	85.71^{++}
MUSCLE 3.7	76.42^{++}	94.04^{+}	87.06^{++}	87.27^{++}	79.71^{+}	84.69^{++}
PROBCONSRNA 1.10	$\mathbf{80.08}^{++}$	$\mathbf{94.48}^{++}$	$\mathbf{88.07}^{++}$	$\mathbf{92.58}^{++}$	$\mathbf{84.76}^{++}$	$\mathbf{87.90}^{++}$

Average sum-of-pair scores (SPS) of the benchmarked programs on BRAliBase II. The The symbols are analogous to Table 6.1.

Tab. 6.9: Column scores on BRAliBase II

Method (DNA)	G2In	rRNA	SRP	tRNA	U5	Total
DIALIGN-TX 1.0	60.85	84.33	70.95	68.05	62.71	69.03
DIALIGN-T 0.2.2	36.51^{--}	50.00^{--}	42.34^{--}	52.01^{--}	50.34^{--}	46.43^{--}
DIALIGN 2.2	60.90^{0}	81.08^{--}	68.53^{--}	67.59^{0}	60.11^{-}	67.29^{--}
CLUSTAL W2	61.24^{0}	86.72^{0}	76.61^{++}	76.20^{++}	65.11^{+}	72.85^{++}
T-COFFEE 5.56	60.24^{0}	82.56^{-}	71.63^{0}	69.23^{0}	62.93^{0}	69.01^{0}
POA V2	55.21^{--}	80.38^{--}	73.77^{++}	66.03^{0}	61.63^{0}	67.12^{--}
MAFFT 6.240 L-INSi	65.23^{+}	87.49^{+}	76.75^{++}	84.59^{++}	68.46^{++}	76.25^{++}
MAFFT 6.240 E-INSi	63.84^{+}	87.34^{+}	76.59^{++}	83.29^{++}	65.71^{++}	75.04^{++}
MUSCLE 3.7	63.20^{0}	87.97^{+}	76.57^{++}	78.01^{++}	64.34^{+}	73.64^{++}
PROBCONSRNA 1.10	$\mathbf{68.70}^{++}$	$\mathbf{88.60}^{++}$	$\mathbf{77.55}^{++}$	$\mathbf{85.46}^{++}$	$\mathbf{71.73}^{++}$	$\mathbf{78.19}^{++}$

Average column scores (CS) of the benchmarked programs on BRAliBase II. The symbols are analogous to Table 6.1.

Tab. 6.10: Run time on BAliBASE 3 and BRAliBase II

Method	Average runtime on BAliBASE 3	Average runtime on BRAliBase II
DIALIGN-TX 1.0	33.37	0.15
DIALIGN-T 0.2.2	27.79	0.08
DIALIGN 2.2	45.41	0.09
CLUSTAL W2	8.72	0.07
T-COFFEE 5.56	315.78	1.95
POA V2	8.07	0.04
MAFFT 6.240 L-INSi	19.51	0.26
MAFFT 6.240 E-INSi	28.26	0.27
MUSCLE 3.7	10.49	0.05
PROBCONS(RNA) 1.12(1.10)	168.65	0.24

Average running time (in seconds) per multiple alignment for sequence families on BAliBASE 3 and BRAliBase II. Program runs were performed on a Linux workstation with a 3.2 GHz Pentium 4 processor and 2 GB RAM.

6.3.2 Results on globally related sequence families

The results of our benchmark on the global alignment databases are listed in the Tables 6.6 and 6.7 for BAliBASE 3 and in Tables 6.8 and 6.9 for core blocks of BRAliBase II. The average CPU times of all methods can be found in Table 6.10. According to the Wilcoxon Matched Pairs Signed Rank Test, DIALIGN-TX outperforms DIALIGN-T 0.2.2, DIALIGN 2.2, POA and CLUSTAL W2 on BAliBASE3 whereby DIALIGN-TX is the only method following the DIALIGN approach that significantly outperforms CLUSTAL W2. Since the methods T-COFFEE, PROBCONS, MAFFT and MUSCLE are focused on global alignments, they significantly outperform DIALIGN-TX on BAliBASE 3. Overall, PROBCONS, MAFFT L-INSi and E-INSi are the superior methods on BAliBASE 3. On BAliBASE 3, the new DIALIGN-TX program is around 1.22 times slower than the previous version of DIALIGN-T and around 1.36 times faster than DIALIGN 2.2.

We get a slightly different picture in the RNA case that we examined using BRAliBase II benchmark database that has an even stronger global character and is the only benchmark database that we used that does not come with core blocks. DIALIGN-TX significantly outperforms POA and all other versions of the DIALIGN approach altough it is still inferior to the global methods CLUSTAL W2, MAFFT, MUSCLE and PROBCONSRNA. The difference between T-COFFEE and DIALIGN-TX on BRAliBase II is quite small, i.e. T-COFFEE outperforms DIALIGN-TX only on the SPS whereas there is no significant difference on the CS. Since MAFFT and PROBCONSRNA have been trained on BRAliBase II the dominance of those methods (especially PROBCONSRNA) is not very surprising. Regarding CPU time DIALIGN-TX is approximately 1.7 times slower than DIALIGN-T 0.2.2 and DIALIGN 2.2 on BRAliBase II.

6.4 Conclusion

In this chapter, we introduced a new optimization algorithm for the segment-based multiple-alignment problem. Since the first release of the program DIALIGN in 1996, a *direct* greedy approach has been used where local pairwise alignments (fragments) are checked for consistency one-by-one to see if they can be included into a valid multiple alignment. In this approach, the order in which fragments are checked for consistency is basically determined by their individual *weight scores*. Some modifications have been introduced, such as *overlap weights* [53] and a more context-sensitive approach that takes into account the overall significance of the pairwise alignment to which a fragment belongs [75]. Nevertheless, a *direct* greedy approach is always sensitive to spurious pairwise random similarities and may lead to alignments with scores far below the possible optimal score (e.g. [80, 56], Pöhler and Morgenstern, unpublished data).

The optimization method that we have introduced herein is inspired by the so-called *progressive* approach to multiple alignment introduced in the 1980s for the classical multiple-alignment problem [26]. We adapted this alignment strategy to our segment-based approach using an existing graph-theoretical optimization algorithm and combined it with our previous *direct* greedy approach. As a result,

we obtain a new version of our program that achieves significantly better results than the previous versions of the programs, DIALIGN 2 and DIALIGN-T.

To test our method, we used standard benchmark databases for multiple alignment of protein and nucleic-acid sequences. Since these databases are heavily biased toward *global* alignments, we also used a benchmark database with simulated local homologies. The test results on this data confirm some of the known results on the performance of multiple-alignment programs. On the globally related sequence sets from BAliBASE and BRAliBase, the segment-based approach is outperformed by classical, strictly global alignment methods. However, even on these data, we could achieve a considerable improvement with the new optimization algorithm used in DIALIGN-TX. On the simulated local homologies, our method clearly outperforms other alignment approaches, and again the new algorithm introduced in this chapter achieved significantly better results than older versions of DIALIGN. Among the methods for global multiple alignment, the program MAFFT [43, 42] performed remarkably well, not only on globally related sequences, but also on locally related ones.

7. CONCLUSION

In this thesis we elaborated on the segment-based approach, which is a promising alternative method for solving the multiple sequence alignment problem since segment-based methods are very suitable for detecting locally conserved motifs regardless of whether they involve only some or all of the input sequences. The major challenge in the segment-based approach is finding the fragments, assigning suitable weight scores to them and assembling those fragments into a multiple sequence alignment such that the resulting alignment depicts the relevant *biological* structure.

We see from the results of DIALIGN-T in chapter 3 that the quality of the input fragments and the quality of the method for assembling them both play a crucial role in the quality of the overall alignment when applying the segment-based approach. As a major focus, we looked in chapter 4 at the assembling method theoretically and proved that, under reasonable assumptions, it is a fixed parameter tractable (FPT) problem in the number of sequences. Since there are various biological domains that influence the structure of the input data, we developed an algorithmic framework in chapter 5 for systematically developing optimal or near-optimal heuristics to solve the problem of assembling an alignment from a given set of input sequences *specific* to the structural nature of the input data. This framework especially inspired the development of DIALIGN-TX as a further improvement of DIALIGN-T (chapter 6). DIALIGN-TX combines the progressive strategy, comparable to global methods like CLUSTAL W and T-COFFEE, and the original greedy strategy of DIALIGN 2.2 and DIALIGN-T. In order to benchmark DIALIGN-TX, we used the BRAliBase II database for DNA sequences and the BAliBASE database for protein sequences for global alignments - both are quite standard for such benchmarks. For local alignments, i.e. to measure the ability of a program to find locally isolated homologous regions, we created the IRMBASE for proteins and DIRMBASE for DNA sequences by arbitrarily implanting artifically generated highly related motifs in otherwise unrelated random sequences. Our benchmarks showed that the recent version of DIALIGN-TX outperforms all other methods on local alignments while still performing very well on global alignments, e.g. it outperforms the very popular global method CLUSTAL W on BAliBASE 3.

In contrast to our focus on improving the assembling of the fragments in the segment-based approach, we did not look too much at the pairwise alignment phase. Only the exclusion of low-scoring sub-fragments in DIALIGN-T (see chapter 3) affects the pairwise alignment whereby we always used the objective weight score function of DIALIGN 2.2. We observe a bias of the objective function toward local isolated similarities, which negatively impacts the quality of the greedy approach. Of course, this effect is compensated for by the various

improvements especially by the progressive strategy and also the exclusion of the low-scoring sub-fragments in DIALIGN-TX. Therefore, further research should focus on the improvement of the objective function in combination with the assembling method of the resulting fragments. An idea in that direction would be to additionally consider in the objective function the situation in which a fragment occurs. At the moment, the weight score of a fragment f depends only on its length, its substitution-matrix score and the length of the sequences it occurs in. However, it is also worth looking at the distance to the neighbour fragments in the same sequence pair rather than only looking at the sequence length. Such an approach would bring a more *global* character to the objective function and may help to reduce the current weakness of the segment-based approach on highly globally related sequences. This can be done, for instance, by looking at the global context of low scoring fragments and therefrom determining whether it is a spurious match or a fragment of biological relevance: if such a low scoring fragment occurs in a chain together with other low scoring fragments with only small gaps in between, it is more likely that it is not a spurious match and the whole chain is suspected of being biologically meaningful. Overall, we conclude that there is still room for further improvement of the segment-based approach on global alignments by adjusting the objective function in combination with the assembling strategy.

As another concluding remark, already published in [75], it is worth addressing a fundamental limitation of most multi-alignment methods, including DIALIGN-T and DIALIGN-TX: these methods implicitly assume that homologies and conserved motifs occur in the same *relative order* within the input sequences. There are two major reasons for making this assumption. First, an order-preserving multiple alignment that represents homologies by inserting gap characters into the input sequences provides a convenient visualisation of existing homologies. Second – and more importantly – the order-preservation constraint greatly reduces the *noise* created by random similarities. A program that returns *all* detectable local or global similarities among the input sequences without the above ordering constraints would necessarily return many spurious random similarities. To reduce this noise, arbitrary threshold parameters would have to be applied which, in turn, could prevent a program from detecting some of the *real* homologies. With the ordering constraint that is implicitly imposed by most alignment programs, weak homologies can be detected, provided they are order-consistent with other detected similarities, i.e. if they fit into one single output alignment. Many evolutionary events such as insertions, deletions and substitutions preserve the relative ordering among sequence homologies. In this situation, order-respecting alignment methods are, in principle, able to represent all true biological homologies in one multiple alignment. Nevertheless, for distantly related protein families, non-order-preserving events such as duplications or translocations need to be taken into account. Such events play an important role in the comparative analysis of *genomic* sequences, which has become an important area of research in recent years [63]. Some promising algorithms for the multiple alignment of genomic sequences have been proposed that are able to deal with non-order-conserving evolutionary events [66, 9]. Also here, further progress in multiple protein alignment can be expected if these ideas are applied to protein alignment algorithms.

Altogether, the resulting programs DIALIGN-T and DIALIGN-TX of this the-

sis have been extensively used in molecular biology research, e.g. [38, 30, 89, 61, 68, 2, 16, 67, 65], providing strong evidence that the segment-based approach of computing multiple sequence alignments is a very relevant method and therefore should also be worked on more in future. We showed that the segment-based method implemented in DIALIGN-TX provides very good results on local alignments and by improving it in the future on the objective function in combination with the assembling method the qualitative gap between it and the *purely global methods* like T-COFFEE, PROBCONS and MAFFT on *global* alignments is very likely to be closed, giving us an alignment framework of a very universal character for computing *general* and *(biological) domain-specific* multiple sequence alignments.

BIBLIOGRAPHY

[1] S. Abdeddaïm and B. Morgenstern. Speeding up the DIALIGN multiple alignment program by using the 'greedy alignment of biological sequences library' (GABIOS-LIB). *Lecture Notes in Computer Science*, 2066:1–11, 2001.

[2] C.L. Afonso, E.R. Tulman, G. Delhon, Z. Lu, G.J. Viljoen, D.B. Wallace, G.F. Kutish, and D.L.Rock. Genome of crocodilepox virus. *J Virol.*, 80(10):4978 – 4991, 2006.

[3] E. Althaus, A. Caprara, H.-P. Lenhof, and K. Reinert. Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics. *Bioinformatics*, 18(Suppl. 2):S4–16, 2002.

[4] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene M. Myers, and David J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[5] J. Berg, J. Tymoczko, and L. Stryer. *Biochemistry.* W.H. Freeman and Company, 2002.

[6] M.P. Berger and P.J. Munson. A novel randomized iterative strategy for aligning multiple protein sequences. *Comput. Appl. Biosci.*, 7:479484, 1991.

[7] H.J. Böckenhauer and D. Bongartz. *Algorithmische Grundlagen der Bioinformatik: Modelle, Methoden und Komplexität.* B.G. Teubner, Wiesbaden, DE, 2003.

[8] Michael Brudno, Michael Chapman, Berthold Göttgens, Serafim Batzoglou, and Burkhard Morgenstern. Fast and sensitive multiple alignment of large genomic sequences. *BMC Bioinformatics*, 4:66, 2003.

[9] Michael Brudno, Sanket Malde, Alexander Poliakov, Chuong B. Do, Olivier Couronne, Inna Dubchak, and Serafim Batzoglou. Global alignment: finding rearrangements during alignment. *Bioinformatics*, Suppl. 1:i54–i62, 2003.

[10] Michael Brudno, Rasmus Steinkamp, and Burkhard Morgenstern. The CHAOS/DIALIGN WWW server for multiple alignment of genomic sequences. *Nucleic Acids Research*, 32:W41–W44, 2004.

[11] Kenneth L. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16:23–25, 1983.

[12] Eduardo Corel, Ramzi El Fegalhi, Fanny Gérardin, Mark Hoebeke, Marc Nadal, Alexander Grossmann, and Claudine Devauchelle. Local similarities and clustering of biological sequences: New insights from N-local decoding. In *The First International Symposium on Optimization and Systems Biology*, pages 189–195, Beijing, China, 2007.

[13] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, USA, 2001.

[14] Florence Corpet. Multiple sequence alignment with hierarchical clustering. *Nuc. Acids Research*, 16:10881–10890, 1988.

[15] M.D. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 6:345–362, 1978.

[16] G. Delhon, E.R. Tulman, C.L. Afonso, Z. Lu, J.J. Becnel, B.A. Moser, G.F. Kutish, and D.L. Rock. Genome of invertebrate iridescent virus type 3 (mosquito iridescent virus). *J Virol.*, 80(17):8439 – 8449, 2006.

[17] E. Depiereux and E. Feytmans. Match-box: a fundamentally new algorithm for the simultaneous alignment of several protein sequences. *CABIOS*, 8:501–509, 1992.

[18] Gilles Didier, Ivan Laprevotte, Maude Pupin, and Alain Hénaut. Local decoding of sequences and alignment-free comparison. *J Computational Biology*, 13:1465–1476, 2006.

[19] C.B. Do, M. Brudno, and S. Batzoglou. ProbCons: probabilistic consistency-based multiple alignment of amino acid sequences. In *Proceedings Nineteenth National Conference on Artificial Intelligence*, pages 703–708, 2004.

[20] Chuong B. Do, Mahathi S.P. Mahabhashyam, Michael Brudno, and Serafim Batzoglou. ProbCons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15:330–340, 2005.

[21] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[22] D. Driscoll and P. Copeland. Mechanism and regulation of selenoprotein synthesis. *Annu. Rev. Nutr.*, 23, 2003.

[23] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis*. Cambridge University Press, Cambridge, UK, 1998.

[24] P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, 1994.

[25] R.C. Edgar. MUSCLE: Multiple sequence alignment with high score accuracy and high throughput. *Nuc. Acids. Res.*, 32:1792–1797, 2004.

[26] D. F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, 25:351–360, 1987.

[27] Paul P. Gardner, Andreas Wilm, and Stefan Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Res.*, 33:2433–2439, 2005.

[28] C. Gille. Structural interpretation of mutations and SNPs using STRAP-NT. *Protein Sci*, 15(1):208–210.

[29] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum flow problem. *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, 1986.

[30] T. Golubchick, M.J. Wise, S. Easteal, and L.S. Jermiin. Mind the gaps: Evidence of bias in estimates of multiple sequence alignments. *Mol. Biol. Evol.*, 24(11):2433 – 2442, 2007.

[31] O. Gotoh. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162:705–708, 1982.

[32] O. Gotoh. Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Comput. Appl. Biosci.*, 9:361370, 1993.

[33] O. Gotoh. A weighting system and algorithm for aligning many phylogenetically related sequences. *Comput. Appl. Biosci.*, 11:534–551, 1995.

[34] O. Gotoh. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *J. Mol. Biol.*, 264:823–838, 1996.

[35] C. Grasso and C. Lee. Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. *Bioinformatics*, 20:1546 – 1556, 2004.

[36] R. Guigó, Pankaj Agarwal, Josep F. Abril, Moisés Burset, and James W. Fickett. An assessment of gene prediction accuracy in large DNA sequences. *Genome Research*, 10:1631–1642, 2002.

[37] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, Cambridge, UK, 1997.

[38] J. Hamazaki, S. Iemura, T. Natsume, H. Yashiroda, K. Tanaka, and S. Murata. A novel proteasome interacting protein recruits the deubiquitinating enzyme UCH37 to 26S proteasomes. *The EMBO Journal*, 25:4523 – 4536, 2006.

[39] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89:10915–10919, 1992.

[40] D.G. Higgins and P.M. Sharp. Clustal - a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73:237–244, 1988.

[41] Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18:314–343, 1975.

[42] K. Katoh, K. Kuma, H. Toh, and T. Miyata. MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nuc. Acids Research*, 33:511 – 518, 2005.

[43] K. Katoh, K. Misawa, K. Kuma, and T. Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nuc. Acids Research*, 30:3059 – 3066, 2002.

[44] J.D. Kececioglu. The maximum weight trace problem in multiple sequence alignment. In *CPM '93: Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, pages 106–119, London, UK, 1993. Springer Verlag.

[45] M. Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, 1983.

[46] Timo Lassmann and Erik L.L. Sonnhammer. Quality assessment of multiple alignment programs. *FEBS Letters*, 529:126–130, 2002.

[47] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–14, 1993.

[48] Christopher. Lee, Catherine. Grasso, and Mark. F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[49] H.-P. Lenhof, B. Morgenstern, and K. Reinert. An exact solution for the segment-to-segment multiple sequence alignment problem. *Bioinformatics*, 15:203–210, 1999.

[50] M. Middendorf. More on the complexity of common superstring and supersequence problems. *Theoretical Computer Science*, 125(2):205–228, 1994.

[51] B. Morgenstern. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15:211–218, 1999.

[52] B. Morgenstern. DIALIGN: Multiple DNA and protein sequence alignment at BiBiServ. *Nuc. Acids Res*, 33, 2003.

[53] B. Morgenstern, A. Dress, and T. Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci. USA*, 93:12098–12103, 1996.

[54] Burkhard Morgenstern. A space-efficient algorithm for aligning large genomic sequences. *Bioinformatics*, 16:948–949, 2000.

[55] Burkhard Morgenstern. A simple and space-efficient fragment-chaining algorithm for alignment of DNA and protein sequences. *Applied Mathematics Letters*, 15:11–16, 2002.

[56] Burkhard Morgenstern, Sonja J. Prohaska, Dirk Pöhler, and Peter F. Stadler. Multiple sequence alignment with user-defined anchor points. *Algorithms for Molecular Biology*, 1:6, 2006.

[57] Burkhard Morgenstern, Oliver Rinner, Saïd Abdeddaïm, Dirk Haase, Klaus Mayer, Andreas Dress, and Hans-Werner Mewes. Exon discovery by genomic sequence alignment. *Bioinformatics*, 18:777–787, 2002.

[58] Burkhard Morgenstern, Nadine Werner, Sonja J. Prohaska, Rasmus Steinkamp Isabelle Schneider, Amarendran R. Subramanian, Peter F. Stadler, and Jan Weyer-Menkhoff. Multiple sequence alignment with user-defined constraints at GOBICS. *Bioinformatics*, 21:1271 – 1273, 2005.

[59] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[60] C. Notredame, D. Higgins, and J. Heringa. T-Coffee: a novel algorithm for multiple sequence alignment. *J. Mol. Biol.*, 302:205–217, 2000.

[61] A. Pavlicek and J. Jurka. Positive selection on the nonhomologous end-joining factor cernunnos-xlf in the human lineage. *Biology Direct*, 1:15, 2006.

[62] William R. Pearson and David J.Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, 85:2444–2448, 1988.

[63] Daniel A. Pollard, Casey M. Bergman, Jens Stoye, Susan E. Celniker, and Michael B. Eisen. Benchmarking tools for the alignment of functional noncoding DNA. *BMC Bioinformatics*, 5:6, 2004.

[64] G.P.S. Raghava, Stephen M.J. Searle, Patrick C. Audley, Jonathan D Barber, and Geoffrey J Barton. OXBench: A benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics*, 4:47, 2003.

[65] B. Ramírez-Zavala and Á. Domínguez. Evolution and phylogenetic relationships of apses proteins from hemiascomycetes. *FEMS Yeast Research*, 8(4):511 – 519, 2008.

[66] Benjamin Raphael, Degui Zhi, Haixu Tang, and Pavel Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, 14:2336 – 2346, 2004.

[67] P.A. Reeves, Y. He, R.J. Schmitz, R.M. Amasino, L.W. Panella, and C.M. Richards. Evolutionary conservation of the flowering locus c-mediated vernalization response: Evidence from the sugar beet (beta vulgaris). *Genetics*, 176(1):295 – 307, 2007.

[68] D.N. Richardson, M.P. Simmons, and A.S.N. Reddy. Comprehensive comparative analysis of kinesins in photosynthetic eukaryotes. *BMC Genomics*, 7:18, 2006.

[69] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[70] D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[71] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[72] P.H.A. Sneath and R.R. Sokal. *Numerical Taxonomy*. Freeman, San Francisco, 1973.

[73] Jens Stoye, Dirk Evers, and Folker Meyer. Rose: Generating sequence families. *Bioinformatics*, 14:157–163, 1998.

[74] A.R. Subramanian, M. Kaufmann, and B. Morgenstern. DIALIGN-TX: greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms for Molecular Biology*, 3:6, 2008.

[75] A.R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. DIALIGN-T: An improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics*, 6:66, 2005.

[76] William R. Taylor. A flexible method to align large numbers of biological sequences. *J. Mol. Evol.*, 28:161–169, 1988.

[77] J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[78] Julie D. Thompson, Frédéric Plewniak, and Olivier Poch. BAliBASE: A benchmark alignment database for the evaluation of multiple sequence alignment programs. *Bioinformatics*, 15:87–88, 1999.

[79] Julie D. Thompson, Frédéric Plewniak, and Olivier Poch. A comprehensive comparison of protein sequence alignment programs. *Nucleic Acids Research*, 27:2682–2690, 1999.

[80] Holger Wagner, Andreas Dress, and Burkhard Morgenstern. Stability of multiple alignments and phylogenetic trees: An analysis of ABC-transporter proteins. *Submitted*.

[81] I.M. Wallace, O. O'Sullivan, D., and Higgins C. Notredame. M-Coffee: combining multiple sequence alignment methods with T-Coffee. *Nucleic Acids Research*, 34:1692–1699, 2006.

[82] Ivo Van Walle, Ignace Lasters, and Lode Wyns. Align-m - a new algorithm for multiple alignment of highly divergent sequences. *Bioinformatics*, 20:1428 – 1435, 2004.

[83] Ivo Van Walle, Ignace Lasters, and Lode Wyns. SABmark - a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, 21:1267 – 1268, 2005.

[84] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1:337–348, 1994.

[85] Michael S. Waterman. Multiple sequence alignment by consensus. *Nuc. Acids Research*, 14:9095–9102, 1986.

[86] J.D. Watson and F.H.C. Crick. A structure for deoxyribose nucleic acid. *Nature*, 171, 1953.

[87] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[88] Andreas Wilm, Indra Mainz, and Gerhard Steger. An enhanced RNA alignment benchmark for sequence alignment programs. *Algorithms Mol Biol*, 2:19, 2006.

[89] W. Yang, P. Ng, M. Zhao, T.K.F. Wong, S.M. Yiu, and Y.L. Lau. Promoter-sharing by different genes in human genome - CPNE1 and RBM12 gene pair as an example. *BMC Genomics*, 9:456, 2008.

INDEX