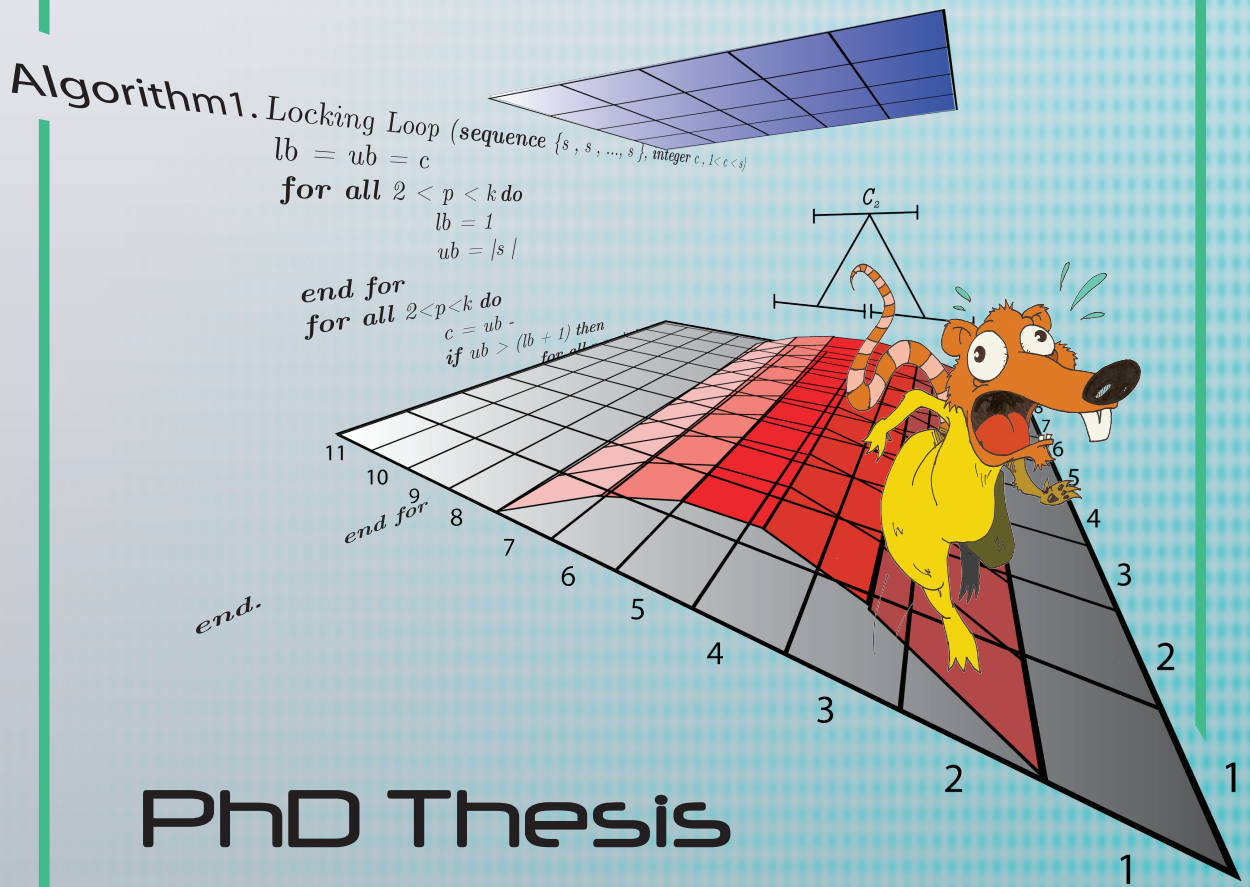
Integrated Multiple Sequence Alignment



PhD Thesis

Michael Sammeth

INTEGRATED MULTIPLE SEQUENCE ALIGNMENT

By

Michael Sammeth

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR RERUM NATURALIUM
AT
TECHNICAL FACULTY
BIELEFELD UNIVERSITY
BIELEFELD, GERMANY
SEPTEMBER 2005

Salud, dinero y amor.

Table of Contents

Abstract

The thesis presents enhancements for automated and manual multiple sequence alignment: existing alignment algorithms are made more easily accessible and new algorithms are designed for difficult cases.

Firstly, we introduce the QAlign framework, a graphical user interface for multiple sequence alignment. It comprises several state-of-the-art algorithms and supports their parameters by convenient dialogs. An alignment viewer with guided editing functionality can also highlight or print regions of the alignment. Also phylogenetic features are provided, e.g., distance-based tree reconstruction methods, corrections for multiple substitutions and a tree viewer. The modular concept and the platform-independent implementation guarantee an easy extensibility.

Further, we develop a constrained version of the divide-and-conquer alignment such that it can be restricted by anchors found earlier with local alignments. It can be shown that this method shares attributes of both, local and global aligners, in the quality of results as well as in the computation time. We further modify the local alignment step to work on bipartite (or even multipartite) sets for sequences where repeats overshadow valuable sequence information. In the end a technique is established that can accurately align sequences containing eventually repeated motifs.

Finally, another algorithm is presented that allows to compare tandem repeat sequences by aligning them with respect to their possible repeat histories. We describe an evolutionary model including tandem duplications and excisions, and give an exact algorithm to compare two sequences under this model.

Preamble

It was a dark night in Barcelona (yes, also in the City of Sun there are dark nights). I was returning home from a larger night session with some problems that are presented in this thesis as Paper IV. My brain felt like spaghetti and my mind still was iterating some recursions with which I hoped to get relations on cascaded duplication histories of tandem repeats. Unaware of my environment, I crossed the tiny alleys of Born, passing the crowded place in front of the Santa Maria del Mar and finally I reached the carrer princesa.

When I arrived home, my flatmate was in front of the TV. From a look in my face he realized that I had been working hard.

– Everything alright? – he asked me.While getting a beer, I mumbled something about sequences, repeats and NP-hard problems.

– What do you mean by *NP-hard*? – my flatmate wanted to know. He was studying architecture, so probably he never had heard of problem complexity classes before. Sipping my beer I tried to outline what is a polynomial time complexity and that non-polynomial algorithms mostly often do not terminate within a reasonable time. I concluded that the design of solutions for NP-hard problems is extremely difficult since every little step that is too much can make the time effort explode.

– Come down, Micha! – he said. – NP-problems is by far not the worst thing that can happen to you. You can desperate one day you find a *NPI-problem*!

I told him that I never in my life had heard of a problem class called NPI.

– I tell you.– my flatmate explained with a content expression on his face. – NPI are the problems that are so big, that you have *ni puta idea* how to solve them.

Acknowledgements

First and foremost I would like to thank Professor Jens Stoye, my supervisor, who always had an ear for my problems and a valuable hint on how to solve them. With his patient guidance and infinite seeming knowledge he demonstrated what it is that makes a doctoral father. Moreover, he is a scientist by heart and one can be always sure of his full support especially when because of approaching deadlines work is going for the nights. Also in private life he is a very nice person and I exemplary want to thank him for his aid in the how-to-get-my-bed-to-Spain problem. I am sure that our collaboration will not stop with the defense of this thesis.

I am also very thankful to Professor Dag Harmsen from the Universitary Hospital in Münster who supported my work with creative suggestions in countless phone calls; luckily now Skype^{TM} has been invented – a program to phone for free over the internet. In joined projects he showed humor and patience with my sometimes chaotic time schedules whenever I had too many projects running simultaneously. On many conferences in Europe or the U.S.A. he introduced me to interesting people and on social events he always knew how to combine good science with a good beer.

Furthermore, I thank Professor Jaap Heringa from the Vrije Universiteit for his ideas during my research period in Amsterdam. Further thanks go to Professor Burkhard Morgenstern from the University of Göttingen for answering all my questions concerning Dialign. He also kindly offered help in an accident with the financing plan at the beginning of the thesis. For the same reason I also would like to thank Professor Ellen Baake from the Bielefeld University. And of course I am very thankful to the Ernst Schering Research Foundation for the doctoral fellowship they gave me. Especially I want to thank Dr. Monika Lessl and Ines Stöhr who also got in contact with the chaos of my tight time plan.

Next, I address some thanks to my research group, the Genome Informatics and our junior resarch group Informatics for Mass Spectronomy in Bielefeld: Heike Samuel

for all organizational help during the thesis; Conni Bannert, Kim Rasmussen and Klaus Schürmann, who are like me longest with the group, for the first German-Danish beer contest and for many funny evenings afterwards; Dr. Sebastian Böcker for good explanations during phylogenetic projects and for demonstrating (together with Dr. Sven Rahmann, Dr. Zsusa Lipták and Mick Kaltenbach) that mathematicians are very funny people; finally I thank all newer members for keeping up the good spirit of the group. Further thanks go to the staff of the Bielefeld University, to the HelpDesk (exemplary Achim Neumann and Peter Serocka) for technical support, to the administration staff of the Technical Faculty (especially to Elke Grütz and to Anke Weinberger), to Folker Meyer for an interesting collaboration, to the Genome Research group (especially Martin Sagasser and Bernd Weisshaar) for providing an eMac while developing the core of QAlign2 and to Thasso Griebel, Malte Brinkmeyer and Felix Tille for productive joined programming.

I thank the groups I visited and the staff of the host universities. People from the Centre for Integrative Bioinformatics at the Vrije Universiteit: Ilse Thomson (for organizing housing and opening my office when I forgot my key), the people from the HelpDesk, Radek Szklarczyk (for being a funny office mate), Victor Simossis (for the funny poker evening) and Jens Kleinjung (for his couch when I still did not have a room in Amsterdam).

From the Genome Bioinformatics Laboratory of the IMIM/CRG/UPF in Barcelona: Maite Cebrián and Alicia de la Vega Gómez (for getting me set up here), Oscar Gonzáles Blanco and Alfons Gonzáles Pauner (for not getting upset when I was doing chaotic things with their systems), Dr. Eduardo Eyras and Professor Roderic Guigó (for the patience with the interrupts in my work for our project), Dr. Tyler Alioto (for writing me a German evaluation of my English), Dr. France Denoeud (especially for last minute corrections on this manuscript), Dr. Jan-Jaap Wesselink (for having a different but still quite good taste of beer), Julien Lagarde (for being French), Dr. Sylvain Foissac (for the 1001 jokes he told), Francisco Cámara Ferreira (for funny evenings), Dr. Josep Abril Fernando (for technical support while printing a poster) and Dr. Mar Albà (for fruitful discussions).

Finally, I wish to thank the following: Laura (for being like she is), my family (especially my mother), my former fellow students from Würzburg: Alex, Alexa, Ali, Andi, Anna, Claudius, Dirk, Dorle, Jordi, Kerstin, Kori, Ruppi, Tobi (for ongoing nice chats and visits), friends from Bielefeld (Arnau, Astrid, Defne, Diego, Elena, Esra, Fernando, Eva, Jorge, Juan, Luis, Laura, Martin, Nana, Pippi, Roberto, Serena,

Silke, Soraya, Las Tapas, Tero), from Amsterdam (Anjali, Anna, Bernat, Christina, Francesco, Jean-Lucca, Jose, Julia, Luis, Marie, Maxie, Nacho, Paz, Pep, Sergio, Steve) and from Barcelona (Alfonso, Bet, Charles, Enrique, Jorge, Montse, Miguel, Pat, Pep, Ramon, Roberto, Rut, els WEKEs).

Barcelona, Catalunya — Michael Sammeth
August 25, 2005

Chapter 1

Introduction

1.1 Summary

In this thesis, we focus on sequence analysis in general and multiple sequence alignment in particular. *Multiple sequence alignment* – although already investigated for over 20 years now – is still a central subject of sequence comparison. Most problems in multiple alignment can be reduced to the fact that the multiple alignment problem is known to be NP hard (Section 2.3). Therefore, a series of heuristical methods have been developed, a hand full of which are included in the current list of state-of-the-art algorithms, e.g. [58] [81] [65] (Section 2.4). However, benchmarks demonstrated efficiently that even these algorithms fail to align certain problems correctly [92] [47] [74] [68] (Section 2.5).

One major problem that we tackle in this thesis is the lack of a *free and open graphical environment* for sequence analysis. Although many bioinformatical techniques to compare and analyze sequences have been elaborated, the biological expert often is prevented from using them by technical problems. We implemented the QAlign package that primarily is designed as a graphical platform for alignment generation, modification and comparison [75]. The package provides a series of visualization features and recently also tools to perform phylogenetic analyzes have been added. QAlign is written in system-independent and object-oriented Java and the modular structure guarantees an easy extensibility. The program is freely

available over the internet and we are in the process to make it also open source (`http://gi.cebitec.uni-bielfeld.de/QAlign`).

Further, it is a well known fact that the strengths of alignment techniques currently available differ substantially [47] [74] [68]. This is a consequence of how the problem of the input is met by the respective objective function and heuristic shortcuts of the algorithms. However, methods combining the benefits of different alignment methods are highly desirable. Two totally different techniques to create global multiple alignment are the divide-and-conquer alignment and the Dialign algorithm. The divide-and-conquer method, based on a branch-and-bound technique with exponential effort w.r.t. the number of input sequences, is constructing an alignment that is (close to) the mathematically optimal sum-of-pair score. On the other hand, Dialign integrates a greedy optimization over local alignments and therefore is more suited for sequences with less global similarity (e.g., data sets with big indels). We worked out a *hybrid algorithm* and demonstrate it to combine the benefits of global and local as well as heuristic and exact multiple alignment (Paper II).

Another problem known to confuse all existing alignment methods is the presence of repeated elements in the sequence set. We aggregate that problem with an algorithm that differentially evaluates sequence similarities in repeated sequence areas and in substrings that are not repeated. In the end, similarities found in both parts are merged according to scores assigned by a modified Dialign objective function. In the *repeat-sensitive alignment method*, repeats can no longer disturb the alignment of information carrying patterns and vice versa (Paper III).

In contrast, sometimes repeats are adding information to the sequence of letters. Sequences that consist of exclusively tandem repeats, for instance, often carry a phylogenetic signal within the micro-variation in the evolution of the repeat copies. Such evolutions are known for a tandem repeat cluster in *Staphylococcus aureus*, a bacterium with growing importance in molecular biomedicine. We describe an evolutionary model with the possibility of taking into account the duplication and excision of tandem repeat copies and present a high-dimensional alignment method to align

tandem repeats w.r.t. their possible repeat histories. In the end, a distance measure can be derived from the alignment score that additionally considers the evolution of the repeat copies (Paper IV).

1.2 Overview

The remainder of this thesis is organized as follows. This chapter further introduces the biological background (Section 1.3) and gives basic definitions of sequence analysis (Section 1.4). In Chapter 2 we introduce a notation for the alignment methods discussed later on. Furthermore, we summarize some popular and state-of-the-art methods for multiple sequence alignment.

Chapter 3 describes the single projects that are part of the thesis. For each project, detailed information about preliminary work in the field is summarized and specific biological backgrounds are introduced as required. In the end, possibilities for improving or refining the developed methods are given.

First we describe the QAlign software, a graphical framework we set up for sequence analysis, especially for multiple alignment and phylogenetic analysis (Paper I). Additionally to a description of the features included in the program, we present some novel features of a recently finished new version QAlign2. Next, we focus on two special alignment algorithms. The first one (Paper II) is a mixture of global sum-of-pairs (divide-and-conquer) alignment and anchors found by segment-to-segment alignment (Dialign). We demonstrate that it shares time characteristics and quality attributes of both underlying methods and thus is suited for data sets with high global affinity and sequences that are more locally related. In addition to information from the publication, an algorithm is given describing how to constrain the divide step to the anchors.

The other alignment algorithm we present is sensitive to repeat boundaries that are submitted with the input (Paper III). A special strategy to collect anchors is applied in order to evaluate the repeated sequence information separately from the

not-repeated substrings. We briefly discuss the optimization for noisy signals. Paper IV finally describes a high-dimensional alignment method that is used to get phylogenetic distances sensitive to duplication histories of tandem repeats. Hitherto we describe also an evolutionary model with in total four operations: excisions, duplications, substitutions and indels. The thesis gives extended information about calculating exact costs for cascaded duplication events.

Chapter 4 puts all contributions in common context and emphasizes parallels as well as differences in intersecting applications. It includes a draft survey of the future applications of contributions as far as we can estimate the time this thesis has been elaborated. Finally we reproduce the papers in chronological order and in the layout of the respective journals.

1.3 Biological Background

This chapter gives a brief overview of the biochemical and genetical mechanisms that are fundamental for sequence analysis as described within this thesis. The terminology used will be introduced in a self-containing but condensed way and the interested reader may get extended information from standard literature such as [85] or [51].

1.3.1 Biochemical Preliminaries

The manual of life is concealed in the chemical composition of information carrying biopolymers. The DNA (desoxyribonucleic acid) being the most famous of those, contains the blueprints of many organisms conserving these information in time while they are passed from generation to generation. A polymerized backbone of sugars joined by phosphates stores the information in *nucleotides*, desoxyribose rings that differ in the base substituent: adenine (`A`), cytosine (`C`), guanine (`G`), and thymine (`T`). `A` and `G` are chemically purines (`R`), while `C` and `T` carry a pyrimidin ring (`Y`). According to the number of hydrogen bonds formed when composing the helix structure, `G` and `C` are strong (`S`) with three H-bonds, whereas `A` and `T` are weak (`W`) bases joined by just two H-bonds. Finally, the side-chains on the bases separate the amino (`M`) bases

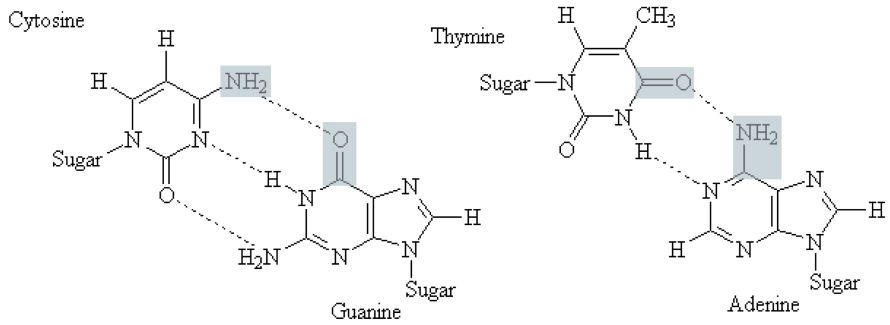


Figure 1.1: Biochemical attributes of the four DNA bases: the 6-edged pyrimidine ring and the purine ring system with 9 edges, the name-giving amino and ketone groups (grey shaded) and the number of H-bonds between respective base pairs (dashed lines).

`A` and `C` from the keto bases `G` and `T` (Figure 1.1).

Throughout the DNA molecule there are protein-coding *genes*, functional units that give rise to the chemically totally different (poly-)peptides or *proteins*. The latter are responsible for the majority of structural or enzymatic (i.e., chemically catalytical) functions in a cell. Proteins are composed of (L-)amino acids, of which about 20 different are known to be used by nature[1]. When transforming the information encoded in genes into proteins (*expression*), RNA (ribonucleic acid) polymers are used as intermediate products. RNA is chemically very similar to DNA, but uses ribose sugars which due to the additional hydroxygen group cannot form such compact and stable helical structures as DNA. Moreover, in RNA uracil (`U`) is used instead of thymine.

The process of copying DNA to RNA is called *transcription*, and the transformation from RNA to proteins is called *translation*. The complete way of information from DNA over RNA to proteins is often called the *genetic flow* of information. Translation can be understood as mapping from nucleotides to amino acids. The rules for mapping nucleotides to amino acids is known as the *genetic code*. Nature uses sets of

[1]Classical genetics knows exactly 20 amino acids. However, there is a growing interest in selenocysteine, a modification of cysteine which sometimes is counted as the 21st amino acid.

First Codon Position	Second Codon Position: U		C		A		G		Third Codon Position
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC	Phe	UCC	Ser	UAC	Tyr	UGC	Cys	C
	UUA	Leu	UCA	Ser	UAA	Stop	UGA	Stop	A
	UUG	Leu	UCG	Ser	UAG	Stop	UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC	Leu	CCC	Pro	CAC	His	CGC	Arg	C
	CUA	Leu	CCA	Pro	CAA	Gln	CGA	Arg	A
	CUG	Leu	CCG	Pro	CAG	Gln	CGG	Arg	G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC	Ile	ACC	Thr	AAC	Asn	AGC	Ser	C
	AUA	Ile	ACA	Thr	AAA	Lys	AGA	Arg	A
	AUG	Met	ACG	Thr	AAG	Lys	AGG	Arg	G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC	Val	GCC	Ala	GAC	Asp	GGC	Gly	C
	GUA	Val	GCA	Ala	GAA	Glu	GGA	Gly	A
	GUG	Val	GCG	Ala	GAG	Glu	GGG	Gly	G

Table 1.1: The standard genetic code used by most organisms of ourdays environment. The first (left), second (top) and third (right) codon position determine a nucleotide triplet that encodes a specific amino acid given in the three-letter-code (body).

three nucleotides to encode the 20 amino acids. Three is the minimum number necessary, since with the encoding capability of four states (i.e., A, C, G, T respectively U), a set of two nucleotides can only cover $4^2 = 16$ amino acids. However, since nucleotide *triplets* can cover in total $4^3 = 64$ codes, the genetic code is *redundant* (Table 1.1).

1.3.2 History of Genetics

Although today the genetic flow of information is well understood and part of the basic training in molecular biology, the way to its discovery was hard. It started when in 1869 *Friedrich Miescher* [56], a physiologist from Basel in Switzerland, firstly isolated DNA. However, it took more than seventy more years (until 1944) before *Oswald Avery*, *Colin MacLeod* and *Maclyn McCarty* identified DNA as the carrier of the genetic information [5]. They triggered a world-wide competition for the closer

investigation of the DNA biopolymer that found its climax in the discovery of the double helix structure by *James Watson* and *Francis Crick* in 1953 [100]. However, a lot of supplementary work by others was necessary to make this discovery possible. For instance very difficult X-ray photos of DNA structures (with lense opening times of more than a hundred hours) by *Rosalind Franklin* are likely to have given a crucial hint [52]. Other observations in the 1930s reported short-living RNA-forms in the cell nucleus, and the in-depth investigation of RNA viruses (e.g., the tobacco mosaic virus) by the american biochemist *Wendell Stanley* (crystallization in 1935 [80]) finally led in 1958 to the formulation of the hypothesis for the genetic flow of information by Francis Crick. Today, this hypothesis is accepted and known as the *central dogma of molecular biology*.

By the discovery of the genetic flow, the sequence composition of proteins already was known since the British biochemist *Fred Sanger* had investigated the sequence of insulin in 1954. However, there still were enough open questions about the transcription and translation processes. A central point was the genetic code revealed independently by *Marshall Nirenberg* [63] and *Kar Khorana* [43] in the early 60ies. By experiments with RNA chains consisting of unique nucleotides (e.g., `UUUUUUUU`) in 1961 they discovered the first *triplets* responsible to encode a specific amino acid; the rest of the genetic code followed until 1965[2].

Raising interest to investigate genes and their expression then required the development of efficient technical methods. In the seventies (1975–1977) first sequencing techniques for DNA molecules were developed by *Frederick Sanger* [77] as well as *Allan Maxam* and *Walter Gilbert* [54]. While Maxam and Gilbert chemically break the DNA strand at the positions of certain nucleotides, Sanger's method is based on producing several copies with controlled polymerization stops after certain nucleotides. In both instances, the fragments whose marginal base is known are then ordered according to their size, allowing to directly derive the base sequence. Although Sanger's

[2]Obviously these *in vitro* translation experiments were only possible with a high salt concentration. Under physiological conditions poly-`U` easily form quadruple helices or other wicked tertiary structures that are impossible to translate.

termination method is still used, today more sophisticated sequencing methods have been established (e.g., Pyrosequencing [66]).

Finally by the mid-eighties another important technique, the *polymerase chain reaction* PCR, was developed by *Kary Mullins* [71, 60]. The PCR *amplifies* a certain area of the DNA molecule, yielding hundreds of copies of the target region. By multiplicating the quantity, it is possible to give visual evidence of reactions that cannot be recognized with one or just a few copies. The method is prune to errors, so not all of the generated copies are necessarily identical: since copying the DNA is a thermodynamic process, it sometimes happens that the polymerase enzyme just "falls off" the DNA, causing a shorter replicate. Another source of differences is the natural error rate of the enzyme (Section 1.4.2). For a detailed description of PCR protocols used to today see [20].

1.4 Sequence Analysis

This section introduces the terminology necessary for sequence analysis in order to describe the algorithms presented in the thesis. We show a basic notation for stringology and biological sequence analysis. Then, we describe events of sequence evolution and assign them mathematically defined operations.

1.4.1 Biological Sequences

In sequence analysis, biopolymers like DNA, RNA or proteins are in general reduced to the sequence of their monomer components. Hitherto, in general a capital letter is assigned to each of the monomers of the respective chemical alphabet, e.g. `A, C, G, T` for DNA (Section 1.3.1). The *alphabet* Σ of a sequence hence is a finite set of letters that normally contains four letters in case of DNA or RNA, or twenty letters for proteins.

Table 1.2 summarizes the extended IUPAC (international union of pure and applied chemistry) notation that knows some additional letters, so called *ambiguities*. For nucleic acids, each pairwise combination of bases is given a symbol according to a

Nucleotide codes

Symbol	Meaning	Origin of designation
A	{A}	adenine
C	{C}	cytosine
G	{G}	guanine
T	{T}	thymine
U	{U}	uracil
R	{A, G}	purine
Y	{C, T, U}	pyrimidine
M	{A, C}	amino
K	{G, T, U}	keto
S	{C, G}	strong interaction
W	{A, T, U}	weak interaction
B	{C, G, T, U}	not-A
D	{A, G, T, U}	not-C
H	{A, C, T, U}	not-G
V	{A, C, G}	not-T, not-U
N	{A, C, G, T, U}	any

Amino acid codes

Symbol	3-letter code	Amino acid
A	Ala	alanine
C	Cys	cysteine
D	Asp	aspartate
E	Glu	glutamine
F	Phe	phenylalanine
G	Gly	glycine
H	His	histidine
I	Ile	isoleucine
K	Lys	lysine
L	Leu	leucine
N	Asn	asparagine
P	Pro	proline
Q	Gln	glutamine
Q	Gln	glutamine
S	Ser	serine
T	Thr	threonine
V	Val	valine
W	Trp	tryptophane
Y	Tyr	tyrosine
X		any amino acid

Table 1.2: Summary of single-letter code recommendations according to the IUPAC standard.

common chemical attribute (`R,Y,S,W,M,K`, Section 1.3.1). Additionally, codes for all combinations of three letters from Σ are introduced, i.e., `B` stands for all bases but `A`, `D` for non-`C` bases, `H` for non-`G`, and `V` for non-`T` or non-`U` nucleotides. A `N` symbol like a wildcard represents all four bases, and correspondingly a `X` stands for one of the 20 amino acids of a protein. The ambiguity codes normally are inserted wherever due to technical shortcomings during the sequencing process (Section 1.3.2) a monomer could not be identified uniquely. Note that Σ may also consist of letters defined for other sequence attributes as described later when introducing macro alphabets (Section 1.4.3).

Formally, a string s of letters from Σ is called a *sequence* over Σ. Sequences in a common set are numbered consecutively $S = \{s_1, s_2, .., s_k\}$ and in pairwise comparisons two arbitrary sequences from S are denoted by s_p and s_q. However, in instances where there are only two sequences present in a set, we sometimes shortly call them s and t. The length of a sequence s, i.e., the number of characters it contains, is denoted by $|s|$. A sequence of length zero is denoted by the empty string symbol ϵ, and $(\Sigma)^+$ denotes the set of all nonempty strings over the alphabet Σ. *Substrings* of consecutive letters $s[x, y]$ are defined by a start position x and an end position y within s, $1 \leq x \leq y \leq |s|$. If only a single position is included in the interval (i.e., $x = y$), the simplified notion $s[x]$ is used instead. Special substrings are *prefixes* ($x = 1$) or *suffixes* ($y = |s|$).

1.4.2 Sequence Evolution

DNA is subject to evolution meaning that due to errors during the replication the sequence information is changed (*mutation*). During the process of *evolution*, the accumulation of mutations often leads to new variants of a molecule with modified attributes and functionality. However, mutations differ substantially in target, molecular mechanism and size of the affected sequence areas. In order to describe them formally, each mutation is assigned a mathematical operation called *edit operation*

by which an original sequence s is transformed into a sequence t, $s \rightarrow t$. The subsequent overview groups mutation events according to the size of the mutated sequence:

Point mutations are changes restricted to a size of up to a few nucleotides. In most instances these errors occur during the replication of the DNA: the copying enzyme is adding a wrong monomer in the thermodynamical process while synthesizing another strand along the parent sequence – in mammal cells this happens normally for about every 10000th position. However, there are other molecular mechanisms that lead to point mutations, like desamination or alkylation of nucleotides and DNA repair mechanisms [85].

Substitutions: formally a letter from the sequence is substituted by another letter from Σ, e.g. `ACTAA` $\leftrightarrow$ `ACTGA`. Substitution operations are treated bidirectionally since we normally do not know which symbol was the original one. Due to differences in the molecular mechanisms, for DNA these mutations are further subdivided into *transitions* (substitutions between chemically equal monomers, purines or pyrimidines) and *transversions* (substitutions of a purine with a pyrimidine or vice versa).

Indels: one or more consecutive letters are inserted into or deleted from the sequence, for instance `AGCGA` $\leftrightarrow$ `AA`. Since *insertion* and *deletion* are directed but inverse operations, the choice depends on the point of view when comparing two sequences and we pool them in a bidirectional operation called indel.

Once the sequence information of the DNA has been changed by a point mutation, the RNA copied from it and possibly also the sequence of translated proteins are altered by the flow of genetic information (Section 1.3.1). However, due to the redundancy in the genetic code (Section 1.3.1), point mutations in the nucleic acids

do not always cause mutations in the derived amino acid sequences. A substitution that is not effecting a change in the translated protein is called a *silent mutation*, wheras all other point mutations are called *effective mutations*.

Rearrangements, in contrast, deal with larger areas of the sequence. If a pattern of the rearranged substrings is known, rearrangements can be limited to the corresponding sequence areas and the operation is of *fixed boundaries*. Otherwise, the boundaries of the operation are *free*, and the rearranged string may start (and end) anywhere within the sequence. Common reasons for rearrangements are splipped-strand mispairing, gene conversion and – in the presence of a multichromosomal genome – unequal recombination (a.k.a. unequal crossing over), but there are also enzyme mediated mechanisms (e.g., transposase-mediated dislocations). Effects of rearrangements can sometimes be quite severe regarding the fact that genes can be dislocated or torn apart.

Duplications result in a duplicate of a substring. Commonly the *tandem duplication* is addressed, where duplicated strings are located directly next to the original: `AGTCAA` $\rightarrow$ `AGTCGTCAA`. Duplications can range from a few bases like in the example (micro-satellites, short tandem repeats) over mini-satellites (10 bp to 100 bp) to large satellites.

Excision: although this term is used for a very heterogeneous group of operations, we use it here for the deletion of an area longer than one position `AGTCAA` $\rightarrow$ `AAA`. Note that usually excisions comprise bigger substrings than shown in the example and therefore are very different from indels. They can be distinguished from large duplications by no similarity between the excised substring and the flanking sequence areas. In coding regions the effects of excisions can be quite severe, e.g., to provoke the illness cystic fibrosis, it suffices to cut out one codon from a gene to corrupt the encoded chloride transporter [25].

Inversions: a substrings is excised and afterwards reinserted in the same genomic place, but in the reverse direction `AGTCAA` → `GGTCAG` (the inversed substring usually is larger). Due to chemical attributes, the double stranded DNA fragment is rotated as whole, and thus the sequence is turned into its reverse complement. Some inversions on the human chromosome X are known to provoke hemophilia type A.

Dislocations: are products of moving a mobile element within the DNA rather than the result of a strand mispairing. `AGTCAA` → `AAGTCA`. Also here the mobile string normally is larger, e.g., 100-400 bp for short interspersed or about 900 bp for long interspersed elements in human. These events are subdivided in *transpositions*, the re-insertion of the element in the same chromosome, and *translocations*, where parts of different chromosomes are exchanged. In contrast to other types of rearrangements, transposable or mobile elements (e.g., L1, *Alu*) need a special structure and the presence of enzymes to change position. Depending on the gene and the position the substring is moved to, effects of dislocations can vary up to carcinomes, muscle distrophia and leukemia.

To describe a certain evolution of a gene or genome, all mutations observed in nature (i.e., accepted muations [21]), more correctly their corresponding edit operations, are combined in an *evolutionary model*. The evolutionary model most sequence comparisons are based on includes only point mutations and therefore is known as the *SI model* (for substions and indels). More recently developed techniques sometimes additionally take into account the possibility of *tandem duplications* and are therefore based on a *DSI model* (duplictations, substions and indels).

1.4.3 Macro Alphabets

As shown in the overview of the different types of mutations (Section 1.4.2), a large part of them is based on substrings rather than on single positions. The mutated sequences can range from a few basepairs up to changes of more than a megabase. To facilitate formal descriptions on techniques, we assign symbols to these substrings. The same symbols can be used to describe substrings with a special function, like motives, regulatory elements, protein domains and repeats. We call the set of all these symbols a *macro alphabet* Σ', since each symbol is actually comprising a range of positions. For Σ' as for Σ holds the attribute of finiteness.

In general we denote a substring with additional information by a *motif* f of its supersequence s, and we denote this relation by $f \subset s$. It is defined by a corresponding start b, an end position e and an additional parameter τ that is capturing the *type* of the fragment $f = (b, e, \tau)$ with $1 \leq b < e \leq |s|$. As for sequences we designate by $|f| := e - b + 1$ the length of the substring that fragment f occupies on sequences. Throughout this thesis, different types τ will be used. They will be assigned capital letters (Figure 1.2) and in sequences we use a slanted font (*A*,*B*,*C*,.., *Z*) to distinguish them from regular protein or nucleotide sequences.

In some classifications of molecular biology, it is necessary to specify *subtypes* [79], [6]. In these instances indices are added to the corresponding types (e.g., $A_1, A_2, ...$). Whenever just the type of a motif f is important, the letter assigned to τ is used as shorthand notation for f, omitting the positional information. Subsequent fragments, e.g. when describing tandem repeats, are annotated as a *macro sequence* s' over Σ'.

a)

```
                    **************************************************
trka_metma      1   MKAVIIGAGEVGYHIAKALSPKNDVVIIDKDEEAAKRADELDVLVIEGNGANAEILS
Q9UXR7          1   MKAVVIGAGEVGYHIAKFLSLTHDVIVVEKDEEVARRADELDVQVLEGNGANADILT
o29420          1   MRIVIAGAGEVGYHLAMSLAPNHDVIIIEKDVSRFERVSELDVVAINGNAANMKVLR

                    **************************************************
trka_metma     57   RV-LQNADLLVAVTGVDEVNIVACMTAKLIMKNKNGWKDTKTIARVSNPDYIDSPVT
Q9UXR7         57   SI-LHDADILIAVTGVDEVNIVACMTAKLILRSHPGWKETKTIARVSNPDYIDVPVT
o29420         57   DAGVERADVFLAVTGNDEVNLLSGLAAKKVG-------AKNVIVRVENPEYVDRPIV

                    **************************************************
trka_metma    113   SRAQVGVDLMICPELALASEVADILSSPSAIDAEMFAEGKVRMTEFAISPESKLVGK
Q9UXR7        113   SRAQVGVDIMICPELALASEVAEVLSSPSAIDAQVFAEGKVQMTEFAISPESKLVGK
o29420        107   KEHPLGYDVLICPQLSLAQEAARLIGIPGAIEVVTFSGGKVEMIELQVMEGSKADGK

                    **************************************************
trka_metma    170   HMQDLKLADCCIVSAVFREDQIIIPHGDDLIKANDHMVVVGKPESMEDLESVFGSKV
Q9UXR7        170   QMQDLKLADCCIVSAVFRDHRIFIPHGSDVIKANDHVVVVGKPEAMESLSSIFGNEV
o29420        164   AIADLYLPQNVVIASIYRNGHIEIPRGDTVLRAGDRVAIVSKTEDVEMLKGIFGPPV

                      **************************************************
trka_metma    226   SHRTRILLIGCGIVGMYLAKLIDKEENADLRIIEHSKSRCIEVAEILENALVLNGDG
Q9UXR7        226   HHKNRVLIIGCGIVGFYLAKLISKEENVDLKIIEHRKDRCIEVAETLEGVLILNGDG
o29420        220   T--RRVTIFGAGTIGSYTAKILAKGMT-SVKLIESSMERCEALSGELEGVRIVCGDA

                    **************************************************
trka_metma    265   TDVSLLREENIEDMDVVVAVTDSDEKNLLCSLLAKQLGAKKVIARADRSDYLPLFEM
Q9UXR7        265   TDLNLLKEENIEDMDVVIAVTDSDEKNLLCALLAKQLGAKKVIARADRSDYLPLFEM
o29420        276   TDIEFLIEEEIGKSDAVLAATESDEKNLLISLLSKNLGARIAIAKVEKREYVKLFEA

                    **************************************************
trka_metma    339   VGIDMAVSPREATVNEVLKLTMGRGIQTLTTIEGERAEIIEYTASEKSRIVGKPLNK
Q9UXR7        339   VGIDMAVSPREATVNEVLKLTMGRGIQTLTTIEGEKAEIIEYTASESSKIVGRPLSK
o29420        329   VGVDVALNPRSVTYNEVSKLLRTMRIETLAEIEG--TAVVEVVVR-NTRLVGKALKD

                    **********************************************
trka_metma    396   VKFPKGAP------------------------------------------
Q9UXR7        396   VKF-----------------------------------------------
o29420        383   LPLPKDAIIGAIVRGNECLIPRGDTTIEYEDRLLVFAKWDEIEKIEEIFR
```

b)

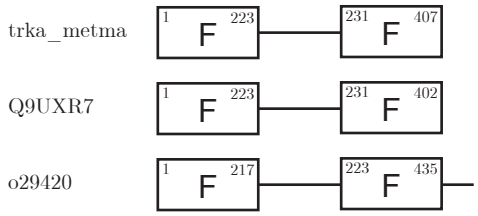


Figure 1.2: Example of reducing the TrkA potassium uptake protein (trka_metma) with two other proteins, each one containing also a KTN NAD-binding domain of type F. (a) shows the alignment of the three amino acid sequences where areas of domains are marked with asterisks. In (b) the condensed schematic representation is given, reducing the protein sequences to their domains (boxes marked with a letter according to the respective τ) and intervening sequences (lines). The example is taken from BAliBase group 6-1a (Section 2.5).

Chapter 2

Multiple Sequence Alignment

In this chapter, we introduce the notation and background relevant for multiple alignment. The *alignment problem* is presented and we show some state-of-the-art methods to solve it. To compare our work with other alignment methods, we give an overview of how to compare the results with existing benchmarks. Finally, phylogenetic analyzes based on multiple alignments are briefly outlined.

2.1 Sequence Alignment

Alignments are a traditional method to compare sequences and to analyze the differences. They compare two (*pairwise alignment*) or more (*multiple alignment*) sequences by searching for a series of individual characters or character patterns that are in the same order in the sequences: identical (*matches*) or similar characters (*mismatches*) are placed in a column, whereas for non-similar positions spacing characters (*indels*, '-') are inserted. The latter are artificially introduced symbols to stretch the sequences in order to model indels and to force related characters in the same column. Therefore, alignments extend the alphabet of the input sequences to $\bar{\Sigma} = \Sigma \cup \{\text{-}\}$. Motivated by notations common in stringology, we denote the length of an alignment $\mathcal{A}$ by $|\mathcal{A}|$ and specify any column of the alignment by $\mathcal{A}[i], 1 \leq i \leq |\mathcal{A}|$. Obviously, $|\mathcal{A}| \geq max|s_i|, 1 \leq i \leq k$.

A *global alignment* of a family of k sequences $S = (s_1, s_2, \ldots, s_k)$ over a finite

alphabet Σ can be represented as a $k \times |\mathcal{A}|$ matrix

$$\mathcal{A}_S = \begin{pmatrix} a_1[1] & a_1[2] & \dots & a_1[|\mathcal{A}|] \\ a_2[1] & a_2[2] & \dots & a_2[|\mathcal{A}|] \\ \dots & & & \\ a_k[1] & a_k[2] & \dots & a_k[|\mathcal{A}|] \end{pmatrix}$$

with entries in the extended alphabet $\bar{\Sigma}$, such that ignoring the spacing characters the pth row reproduces sequence s_p and there is no column consisting exclusively of spacings. A maximal run of adjacent spacing characters is called a *gap*.

Local alignments in contrast align only fragments of the input sequences and are more suitable when comparing sequences that are related in some areas but dissimilar in others. Formally, a local alignment $\mathcal{A}_{f_1,f_2,\dots,f_k}$ is the alignment of fragments $\{f_1, f_2, \dots, f_k\}$ of the input sequences, where $|f_1 \subset s_1, f_2 \subset s_2, \dots, f_k \subset s_k$. In special cases two fragments $f_p = (b_p, e_p, \tau_p)$ and $f_q = (b_q, e_q, \tau_q)$ are aligned *gapless*. Such relations are called *anchors* that can be characterized by $\mathcal{A}_{f_p,f_q} = (b_p, b_q, |f|)$, with alignment length $|f| = |f_p| = |f_q|$. Note that whenever not further specified, the term *alignment* will refer to global alignment throughout this thesis.

2.2 Scoring Alignments

We denote the set of all alignments possible for a data set S by the alignment space Ω_S. In an *optimal* alignment, however, nonidentical characters and gaps are placed to bring as many identical or similar characters as possible into vertical register. To measure similarity between characters, *scoring schemes* have been set up, which roughly can be divided into two groups, cost or similarity scoring schemes. Both define functions that return a numerical value for the atomary comparison of two characters of the alphabet: *costs* impose penalties according to the degree of dissimilarity ($cost(c_1, c_2) : (\bar{\Sigma})^2 \rightarrow \mathbb{R}_0^+$), whereas *similarity* values can either be positive, negative or zero ($sim(c_1, c_2) : (\bar{\Sigma})^2 \rightarrow \mathbb{R}$). Both sorts of scoring schemes can be transformed into each other under certain conditions [78], however, due to some algorithmical advantages we will use exclusively cost schemes throughout this work.

Most cost schemes comply with the attributes of a metric (Section 2.6). The most simple cost scheme is the so called *unit cost*, where each match between two characters $(c_1, c_2) \in \bar{\Sigma}$ is assigned no cost ($cost(c_1, c_2) = 0$ iff $c_1 = c_2$) and mismatches or indels all are assigned the same cost of a positive unit $u > 0$. More elaborated scoring schemes take into account biological observations, e.g. nucleotide matrices that distinguish between transitions and transversions or the PAM (percent accepted mutation) [21, 37] or BLOSUM (blocks substitution matrix) [37] matrices for proteins. They have been extracted from the substitution rates observed in alignments of different sequence sets. PAM values are derived from global alignments of one evolutionary distance and then extrapolated for other distances, while BLOSUM matrices stem from local alignments of proteins with different evolutionary distances (Blocks database).

Definition 2.2.1 (SP score)**.** The sum-of-pairs (SP) score for a multiple alignment $\mathcal{A}_S$ is composed of the scores for each pair of sequences in the alignment. Such a pairwise alignment $\mathcal{A}_{s_p,s_q}$ is composed of the scores for each character tuple of its columns:

$$cost(\mathcal{A}_{s_p,s_q}) = \sum_{i=1}^{|\mathcal{A}|} cost(a_p[i], a_q[i])$$

Note that $cost(\mathcal{A}_{s_p,s_q}$ sometimes also is called the *distance* between the aligned sequences s_p and s_q. The score of $\mathcal{A}_S$ is then the sum for all character tuples (pairwise projections):

$$cost(\mathcal{A}_S) = \sum_{p<q} \sum_{i=1}^{|\mathcal{A}|} cost(a_p[i], a_q[i])$$

Another issue are gap costs: due to definition of a gap (Section 2.1), the adjacent spacing characters should be treated together in the scoring scheme. The *linear gap costs* scheme just sums up the penalties for each spacing character in a gap $a_p[x, y]$, not taking into account the gap as a whole.

$$cost(a_p[x, y],' - - \ldots -') = \sum_{i=x}^{y} cost(a_p[i], -), a_p[i] \neq' -'$$

On the other hand, there is a strategy to add an additional penalty once a gap is initiated to contribute to the mechanism of a deletion (Section 1.4.2), called *affine*

gap costs. These are therefore defined by two parameters, the gap open *go* and the gap extension score $cost(-, c), c \neq' -'$, with the convention that $go > 0$. The cost of a substring $a_p[x, y]$ aligned with a gap is then

$$cost(a_p[x, y]) = go + \sum_{i=x}^{y} cost(-, a_q[i])$$

Finally, due to technical limitations of the sequencing process (Section 1.3.2), *marginal* gaps are often treated different from *internal* gaps. Marginal gaps, indicating that one of the compared sequences is shorter than the other(s), are often technical artefacts (Section 1.3.2). Therefore they are assigned less costs than internal gaps, commonly marginal gaps are not even penalized at all.

2.3 Multiple Alignment Problem

With the definition of cost schemes and assuming the commonly accepted parsimony principle for nature [17][30], the multiple alignment problem can be described as an optimization problem. Modelling the evolution for a set of sequences $S = \{s_1, s_2, \ldots, s_k\}$ with a cost function $cost()$, we search in the alignment space $\Omega_{s_1,s_2,\ldots,s_k}$ for the alignment that minimizes the costs of the aligned characters.

$$\hat{\mathcal{A}} := argmin\{cost(\mathcal{A}) \mid \mathcal{A} \in \Omega_{s_1,s_2,\ldots,s_k}\}$$

The multiple alignment problem has been proven to be NP-complete in three different methods. Wang and Jiang [96] use the SP scoring scheme (Section 2.2) and define the *Sum-of-Pairs Multiple Alignment* (SPMA) problem. They prove the SPMA problem to be NP-complete by reduction from the *Shortest Common Supersequence Problem* [32]. Later on we will see that it corresponds to the shortest-path problem in a high dimensional DP matrix spanned by the input sequences S (Section 2.4).

A more general description has already been given by Kececioglu in 1993 [42]. He formulates the problem as the *Maximum Weight Trace* (MWT) alignment problem in a graph $G = (V, E)$. The letters of the input strings $s_p[i] \in S$ are vertices V of

the graph. Every edge $e \in E$ of the graph has a positive weight w_e and connects two vertices (letters) that belong to different strings. In an alignment $\mathcal{A}$ these two letters connected by e would be placed in the same column. The set of edges incorporated in an alignment $\mathcal{A}$ is called the *trace* of $\mathcal{A}$. The maximum weight trace problem is defined as the problem to maximize the sum of weights w_e of all edges realized by $\mathcal{A}$. Kececioglu proves that the MWT problem contains the SPMA problem and that the MWT problem is NP-complete by reduction from the *Feedback Arc Set Problem* [32].

2.4 Global Alignment Techniques

It is a hard task to find an optimal alignment, the possibilities grow rapidly even for only a pair of sequences: considering all possible matches and indels, for proteins of length 300, about 10^{88} comparisons have to be computed in an exhaustive search [97][1].

Dynamic Programming: The crucial point is when naively iterating all alignments in Ω, the same comparisons have to be performed exponentially many times. However, there are only quadratically many subproblems when comparing each letter from s_p with a letter from s_q. *Dynamic programming* is a general technique in computer science that circumvents the problem of multiply solving the same subproblem by iteratively solving the subproblems sorted by increasing size. Proceeding in the iteration, each subproblem is solved only once, cutting down the necessary comparisons to the number of subproblems. The essence of dynamic programming is Richard Bellman's *Principle of Optimality*:

An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

[1]The exact number of alignments between to sequences of length n is given by $\left(\begin{array}{c} 2n \\ n \end{array} \right) = \frac{(2n)!}{(n!)^2} \simeq \frac{2^{2n}}{\sqrt{\pi n}}$ [98].

This is a self-evident principle in the sense that a proof by contradiction is immediate. Rutherford Aris restates the principle in more colloquial terms:

If you don't do the best with what you have happened to have got, you will never do the best with what you should have had.

In other words, according to Bellman's Principle the subproblems can be derived from each other, and each optimal partial solution contains partial sub-solutions which on their turn are optimal. Therefore, the subproblems for dynamic programming are to be sorted in recursively iterable cascade. For the *pairwise alignment* problem of two sequences s and t, these cascaded intervals are for instance partially aligned prefixes $s[1,i]$ and $t[1,j]$ for all $i = 1,..,|s|$ and $j = 1,\ldots,|t|$ respectively. To store all partial results, we use as data structure an *alignment matrix* $M_{s,t}$ over the compared sequence pair, including an additional row and column for alignment of ϵ:

$$M_{s,t} := (M_{s,t}[i,j]) \text{ where } M_{s,t}[i,j] = cost(\hat{\mathcal{A}}_{\int[\infty,\rangle],\sqcup[\infty,|]}) \text{ for all } (i,j), 0 \leq i \leq |s|, 0 \leq j \leq |t|$$

For linear gap costs, this matrix can be filled straightforwardly by the well-known recurrence usually attributed to Needleman and Wunsch[2] [62] (Figure 2.1). Note that with a slight modification the recurrence can also be used for local alignment [99].

$$M_{s,t}[i,j] := min \begin{cases} M_{s,t}[i-1,j-1] & +cost(s[i],t[j]) \\ M_{s,t}[i-1,j] & +cost(s[i],\text{-}) \\ M_{s,t}[i,j-1] & +cost(\text{-},t[j]) \end{cases} \tag{2.4.1}$$

Obviously, the row and column representing the empty string have to be initialized with multiples of the respective gap costs:

$$\begin{aligned} M_{s,t}[0,0] &:= 0 \\ M_{s,t}[i,0] &:= M_{s,t}[i-1,0] + cost(s[i],\text{-}) \\ M_{s,t}[0,j] &:= M_{s,t}[0,j-1] + cost(\text{-},t[j]) \end{aligned}$$

[2]Actually the original recursion presented by Needleman and Wunsch was cubic in time complexity.

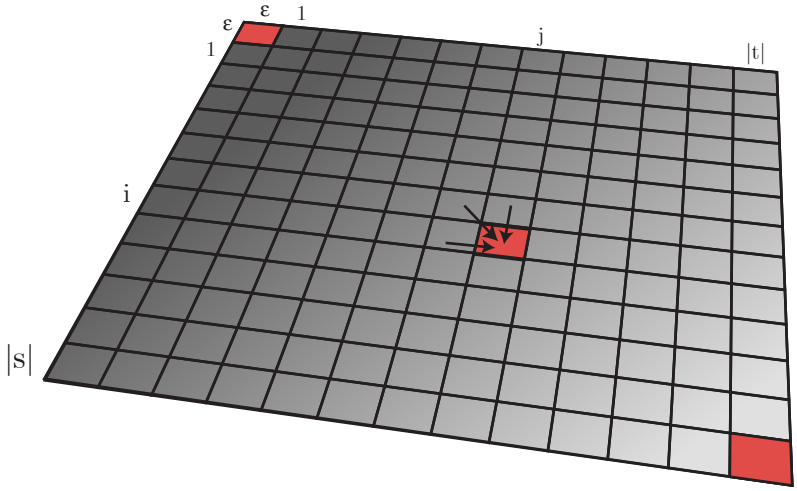


Figure 2.1: A DP matrix for the sequence pair (s, t). Note that the first column and the first row correspond to the empty string ϵ of each sequence. Highlighted are the source cell (upper-left corner) and the sink cell (lower-right corner). An example for an arbitrary sequence pair shows the dependencies when calculating the best score for cell (i, j).

Additionally, we store a pointer from each cell (i, j) in $M_{s,t}$ to the cell(s) that participate immediately (i, j) in the optimal partial alignment(s) $\mathcal{A}_{s[1,i],t[1,j]}$. By this, successively an *alignment graph* is constructed, traversing the matrix from the *source* cell $(0, 0)$ towards the *sink* $(|s|, |t|)$. Iterating the recursion over i and j, the matrix is filled with values. Note that at the same time it is assured that for each recursion depth (i, j) the necessary values of preceding cells $(i-1, j), (i, j-1)$ and $(i-1, j-1)$ are already computed. Once the value for the *sink* cell $(|s|, |t|)$ has been computed, the algorithm terminates with reconstructing the complete alignment graph(s) of the optimal alignment(s) by tracing back the backpointers until the source cell. Obviously the number of problems to be computed by the algorithm are $\mathcal{O}(|s||t|)$, or with n denoting the length of the longest sequence $\mathcal{O}(n^2)$.

Another problem is the calculation of *affine gap costs* when using the dynamic programming technique. Due to the difference between gap open and gap extension costs, when inserting a gap at (i, j), the minimum cost cannot be determined exactly by just taking into account the positions $(i - 1, j)$ and $(i, j - 1)$. Since for these

Bellman's Principle of Optimality is violated, basically one would have to reconsider for each (i, j) all alignment paths that end with a gap, i.e., paths from cells in the ith row until column $j-1$ and from cells in the jth column until row $i-1$. However, Gotoh introduced another method for keeping the time complexity of $\mathcal{O}(n^2)$, but investing three times more memory [33]. His method bases on *history matrices* of the same size as M, that additionally store the values and backpointers of all prefix alignments that end with a gap.

The dynamic procedure can straightforwardly be extended to multiple alignments of k sequences. For these instances, the alignment matrix $M_{\{s_1,s_2,\ldots,s_k\}}$ correspondingly has more dimensions. The effort to fill these *hyperspace* cubes grows exponentially with the number of sequences, generally for k sequences and $n = max\{|s_1|, |s_2|, \ldots, |s_k|\}$, $\mathcal{O}(2^k n^k)$ steps have to be computed. To be specific, n^k partial alignments are to be calculated and for each one $2^k - 1$ originating cells are to be considered. Since the multiple alignment problem is known to be NP-hard (Section 2.3) and dynamic programming retrieves an optimal solution, the exponential effort is inevitable.

However, for affine gap costs the effort of multiple sequence alignment raises to $\mathcal{O}(2^k n^k kn)$. Saving history matrices to preserve the time effort is a bad option, since memory consumption explodes exponentially [3]. In contrast a heuristic on the computation of affine gap costs is used that just considers only the last alignment steps in the hypercube. Altschul demonstrated that the differences between these *quasi-natural* gap costs and the correctly calculated gap costs are marginal [3].

For the exponential costs of multiple sequence alignment, many methods use *heuristics*, shortcuts to compute the result faster. The used heuristics are very heterogeneous and, according to the quality of the results, they are further subdivided in *exact methods* that still guarantee to retrieve the optimal result and *heuristic methods* where this guarantee cannot be given anymore. Here, we show some multiple alignment techniques that are important for the presented work.

Progressive alignments break down the multiple alignment problem to a series of pairwise alignments. Here, iteratively two sequences are aligned until all sequences are aligned. It can easily be seen, that to do so, it is necessary to align a sequence (say t) with a set of already aligned sequences, e.g., $\mathcal{A}_{s_1,s_2,\ldots,s_p}$. For these so-called *profile* alignments, the cost function is adapted to match each position $t[j]$ against all characters in $\mathcal{A}_{s_1,s_2,\ldots,s_p}[i]$ by taking the arithmetic average of the different scores yielded (Equation 2.4.2). The concept is straightforward extendable to the profile alignment of two alignments $\mathcal{A}_{\{s_1,s_2,\ldots,s_p\}}$ and $\mathcal{A}_{\{t_1,t_2,\ldots,t_q\}}$.

$$cost(\mathcal{A}_{\{s_1,s_2,\ldots,s_p\}}[i], t[j]) = \frac{\sum_{u=1}^{p}(cost(a_u[i], t[j]))}{p} \tag{2.4.2}$$

A very crucial point in the progressive alignment strategy is the order in which the sequences are added to the solution. The greedy process never assures to find the optimal result (strong heuristic) and the quality of the result may vary substantially with the alignment order. When difficult alignments are carried out in early stages, the whole method is prune to errors since there is not a lot of sequence information available and misdecisions are easily taken. Difficult alignments here are generally alignments of distant sequences where many gaps and mismatches have to be tolerated to match the sequences. To evade this danger, it is a common technique to compute a quick phylogenetic tree (Section 2.6), a *dendrogram*, to guide the alignment order and align the most related sequences first (Figure 2.2).

Various versions of this basic strategy have been developed. Clustal [90] is one of these strategies that are called *tree alignments*, in contrast to naive progressive methods like *star alignments* that align each sequence against a common consensus sequence. Note that the current version, ClustalW, contains many more algorithmical refinements, e.g., sequence weighting, position-specific gap penalties and it has been trained for the BAliBase benchmark set (Section 2.5, Des Higgins personal communication).

Multiple sequence alignment (MSA) is the slightly unlucky name of a very

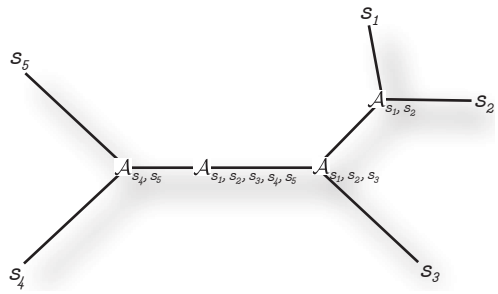


Figure 2.2: A phylogenetic dendrogram to guide the progressive alignment. The input sequences $\{s_1, s_2, s_3, s_4, s_5\}$ are distributed across the leaves of the tree, and internal nodes represent alignments of respective subsets. Guiding trees are usually unrooted and a further progressive step is necessary to align the partial alignments $\mathcal{A}_{\{s_1,s_2,s_3\}}$ and $\mathcal{A}_{\{s_4,s_5\}}$.

efficient implementation for exact multiple alignment [35]. With exponential effort, here the multi dimensional matrix $M_{\{s_1,s_2,\ldots,s_k\}}$ is computed, speeding up the procedure by two exact heuristics.

(i) the *Carrillo-Lipman* heuristic [18]: takes advantage of the score yielded by a quick (i.e., a progressive) alignment $\mathcal{A}_{\{s_1,s_2,\ldots,s_k\}}$ that is precomputed on the sequences. Afterwards, all alignment paths containing cells with a score exceeding the score of the precomputed alignment are excluded from further computations. Obviously the technique can be applied, since $cost(\hat{\mathcal{A}}_{s_1[1,i_1],s_2[1,i_2],\ldots,s_k[1,i_k]})$ can never decrease with growing values $(i_1, i_2, \ldots, i_k)$.

(ii) *homing costs* additionally estimate a lower bound for the alignment costs from the currently computed cell $(s_1[i_1], s_2[i_2], \ldots, s_k[i_k])$ to the sink from the sum of optimal suffix alignments in all pairwise projections, $\sum_{p<q} cost(\hat{\mathcal{A}}_{s_p[i_p,|s_p|],s_q[i_q,|s_q|]})$. A cell $(i_1, i_2, \ldots, i_k)$ for which the homing cost and the sum between the cost of the aligned prefixes $cost(\hat{\mathcal{A}}_{s_1[1,i_1],s_2[1,i_2],\ldots,s_k[1,i_k]})$ together exceed the cost of the heuristic pre-alignment are omitted.

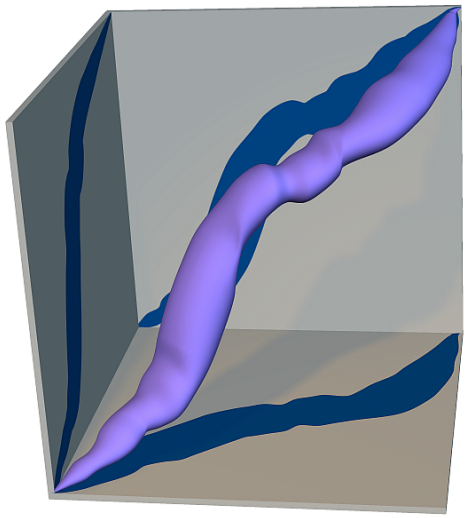


Figure 2.3: In the example the Carrillo-Lipman heuristics and the homing costs cut down the original 3D alignment matrix (grey cube). Only cells that comply with both conditions (i.e., cells within the blue pipe) are regarded for computation of the optimal alignment. Picture rendered by *W. Esser*.

Figure 2.3 shows how the initial search space is narrowed by omitting regions of the alignment graph with edges that fail to pass condition (i) or (ii). Cells of the search space for $\hat{\mathcal{A}}_S$ are stored in a heap structure and further iterated according to the costs assigned to them. Although Gupta *et al.* could not give a proof, they observed that (i) and (ii) omit an overlapping but not necessarily identical set of edges. Without going into details, it is worthwhile knowing that these consistency conditions have been further refined by the *A*Algorithm* [50].

Divide-and-conquer alignment (DCA) [93, 81] is a breaks down the lengths of the aligned sequences. In its common implementation it is used to make the MSA algorithm (conquer step) more efficient: according to a user defined string length L, the algorithm recursively cuts the k input sequences in two parts (Figure 2.4).

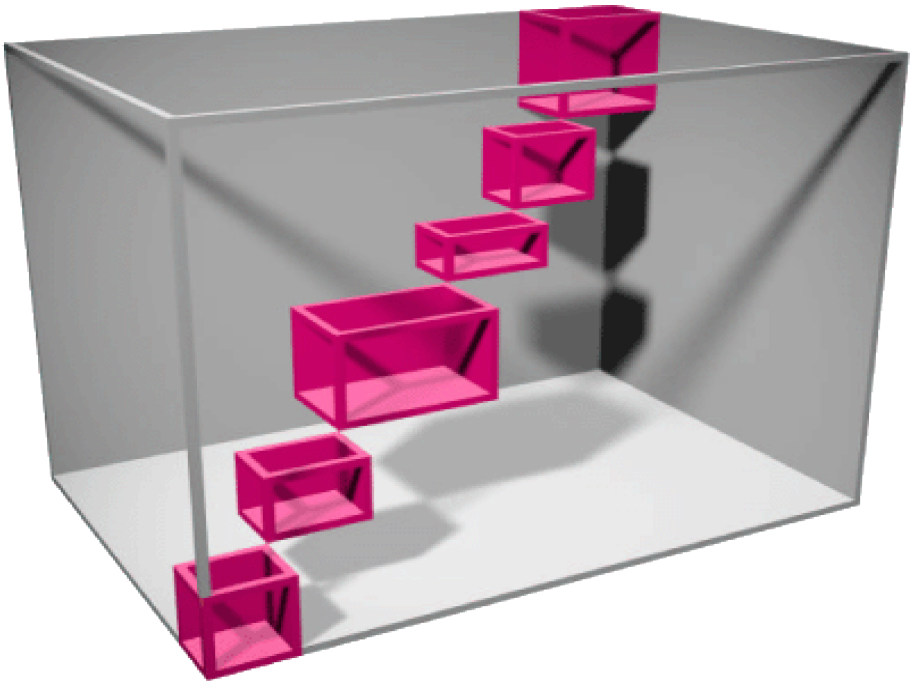


Figure 2.4: Recursively the divide step cuts down the search space of three sequences (grey cube) to subspaces (red boxes). The conquer step then is performed on the subsequences along the borders of these subspaces. Note that the dividing procedure forces the optimal alignment path to contain the cells at the junction points of the subspaces (i.e., where the corners of the red boxes touch each other). Picture rendered by *P. Serocka*.

Of course, the choice of the *cut positions* $\hat{C} = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k\}$ is critical for the success of the DCA procedure, and inadequate cut positions in an early division step can deteriorate the whole alignment. Stoye showed that the heuristics of *minimal additional costs* of pairwise alignment paths in all projected M_{s_p,s_q} yields very good, in many cases optimal cut positions [81]. Hence, this method is often treated as an exact method.

Dialign [58] is a technique to construct multiple alignments from anchors (Section 2.1), which gave the name to the method since in dotplots these anchors appear as diagonals. Possible anchors are collected from the sequence pairs and scored in two

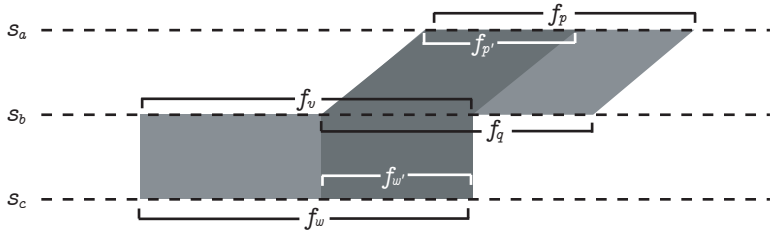


Figure 2.5: Example of two overlapping anchors $\mathcal{A}_{f_p,f_q}$ and $\mathcal{A}_{f_v,f_w}$ (light grey links) on three sequences s_a, s_b and s_c. By the transitive link between positions of s_a and s_c effected by the overlapping area (shadowed dark grey), a third anchor $\mathcal{A}_{f_{p'},f_{w'}}$ is induced.

steps: first they are assigned a *weight* $w(\mathcal{A}_{f_p,f_q})$ according to the probability that the corresponding anchor with length $|f_p| = |f_q|$ and similarity score $sim(f_p, f_q)$ has been found by chance in the input sequences S. To calculate $sim(f_p, f_q)$, for proteins the Blosum62 table is used. An error probability $P(\mathcal{A}_{f_p,f_q})$ that the corresponding anchor has been found by chance is empirically assigned from random sequence experiments. Motivated by objective functions in statistical mechanics the negative logarithm is assigned as weight w:

$$w_{\mathcal{A}_{f_p,f_q}} = -log(P(f_p, f_q)).$$

Later on [57], the scoring function $w()$ has been refined. In a heuristical solution to the multiple alignment problem (Section 2.3), the weights of all overlapping anchors are combined. Two anchors $\mathcal{A}_{f_p,f_q}$ and $\mathcal{A}_{f_v,f_w}$ are considered as overlapping if they share overlapping fragments on the same sequence while the other fragment is situated on different sequences (i.e., $f_q, f_v \subset s_b, f_p \subset s_a, f_w \subset s_c$). In the overlapping region, the character tuples aligned by both anchors are linked *transitively* (Figure 2.5). This transitive correlation implies a third anchor between two hitherto unaligned fragments of the sequences s_p and s_q. The weight of the implied anchor is then used to upweight $w(\mathcal{A}_{f_p,f_q})$ and $w(\mathcal{A}_{f_v,f_w})$ to yield the *overlapping weight* $olw()$:

$$olw(\mathcal{A}_{f_p,f_q}) = w(\mathcal{A}_{f_p,f_q}) + w(\mathcal{A}_{f'_q,f'_u})$$

$$olw(\mathcal{A}_{f_u,f_v}) = w(\mathcal{A}_{f_u,f_v}) + w(\mathcal{A}_{f'_q,f'_u})$$

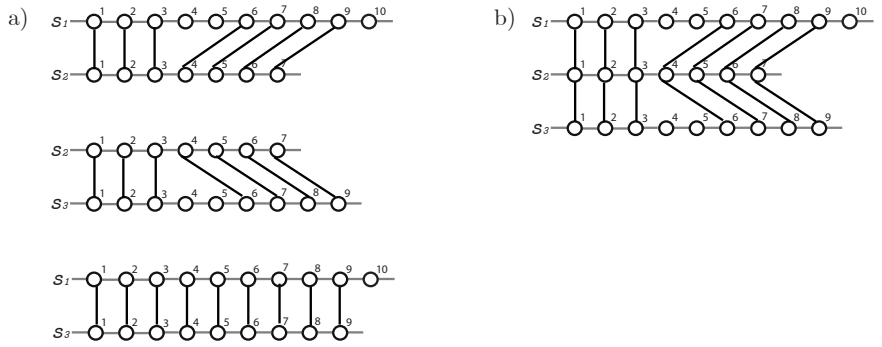


Figure 2.6: Example of the triple comparison for three sequences: (a) the pairwise global alignments for all pairs of s_1, s_2 and s_3; (b) the triple comparison of s_1 against s_3 through s_2. Only positions that are linked in all three sequences receive the combined weight of both pairwise alignments $\mathcal{A}_{s_1,s_2}$ and $\mathcal{A}_{s_2,s_3}$.

In the end, a *greedy* selection adds the anchors one after another to the final solution, as long as they are consistent with each other. *Consistency* here refers to the partial order each anchor imposes on the order of positions in the global alignment $\mathcal{A}_S$.

T-Coffee is another method that incorporates local relationships into global alignment. In the beginning, all pairs of sequences are aligned globally, and additionally the best ten local alignments are collected. Each aligned character pair $(a_p[i], a_q[j]), a_p[i], a_q[j] \neq' -'$, is assigned the score $sim(s_p, s_q)$ or $sim(f_p, f_q)$ score according to the alignment(s) $\mathcal{A}_{s_p,s_q}$, $\mathcal{A}_{f_p,f_q}$ that contained it (*primary library*).

Then these scores are refined w.r.t. the context of the multiple sequences. In another heuristic solution of the multiple alignment problem, here so called *triple comparisons* are performed. To be specific, each set of three sequences is regarded, say a sequence pair s_p and s_q is investigated "through" sequence s_u. If character tuples between s_p and s_q are transitively connected through common alignments with positions in sequence s_u, their signals are enhanced by adding the corresponding alignment scores (Figure 2.6).

Finally, all tuple scores are turned into *position specific scoring matrices* (PSSMs)

that score the alignment of positions rather than of characters, i.e. $cost(s_p[i], s_q[j])$. In the final alignment procedure, a progressive tree alignment, the PSSMs are used instead of the general substitution matrices. Therefore, the method has a tremendous gain in sensitivity for the input sequences.

2.5 Alignment Benchmarking

Since the beginning of multiple alignment research, methods to compare and evaluate alignments have been developed. There are high demands on a good benchmarking method:

Biological significance: the measure of the comparison has to be a biologically meaningful relation of the sequences. In other words, we need the "true" alignment to compare it again the test alignments. Providing gold standards $\hat{A}$ for such comparisons is rather difficult since expert knowledge and in detail analysis of data sets is needed and neither of them can be generated automatically.

Objectivity: the measurement of the comparison should evaluate the alignment methods neutrally and independent of their respective objective function. However, this requires theoretically to develop a new scoring system for every comparison with respect to the OFs used in the compared alignment methods (e.g., SP, maximum likelihood, minimum entropy, star, consensus, etc. [34]). In practice, mostly the popular SP-system based comparisons are used. But we want to stress the fact that this comparison biases slightly against methods not using a SP system to add the signals over multiple sequences. Moreover, it could be demonstrated that some manually elaborated alignments do not optimize the score of any SP scoring scheme [47].

Generality: an ideal benchmark should cover the complete spectrum of alignment problems. Obviously, this is an utopic goal since a finite set including

all alignment problems has not yet been defined. To come closer to statistical stability of the benchmark, experiments have been performed including artificially generated sequences [82, 47]. However, such artificial test sets contradict the requirement for biological significance. In general, alignments are tested with a series of data sets that try to cover a wide range of problems. Another problem that may arise when including too many or too big test sets is that computationally expensive methods may not terminate with generating the necessary alignments in a reasonable time. In practice, there exist two options for these instances: the test data set has to be reduced or the respective alignment algorithms are to be excluded from the benchmark.

2.5.1 Benchmarks based on expert knowledge

McClure. One of the first benchmarks for multiple protein alignment was developed already in 1994 by McClure *et al.* [55]. They composed four data sets of biologically investigated proteins, i.e., data sets of twelve globins, twelve aspartate proteases, twelve kinases and of twelve ribonucleases H. Furthermore, subfamilies of ten respectively six sequences of each data set were defined and benchmarking is performed across all these twelve data sets.

To compare the alignment layouts, the SP scoring system was used and character tuples were scored according to their occurrence in the reference alignment $\hat{\mathcal{A}}$: a character tuple of the test alignment that is also present in the reference is empirically assigned a score of 1; otherwise it is assigned a score of 0. Note that the comparisons are based on positions rather than on characters. The sum over all position tuples then forms the SP-score (Section 2.2).

$$SP_{\hat{\mathcal{A}}}(\mathcal{A}) = \sum_{p<q, 1\leq i\leq|s_p|, 1\leq j\leq|s_q|} \begin{cases} 1 & \text{iff } \mathcal{A} \text{ and } \hat{\mathcal{A}} \text{ align position } i \text{ in } s_p \text{ with } j \text{ in } s_q \\ 0 & \text{otherwise} \end{cases}$$

Since not all positions from a sequence are equally conserved, McClure used the biological background knowledge to restrict the evaluation to functional sequence positions, the so-called *core* of the sequences. Here, the alignment of the positions

is clearly identified and can be given as a reference gold standard. The results then may be given as absolute SP scores over all core positions or as a relative fraction of correctly aligned core positions from the reference.

BAliBase. Five years later, another protein benchmark test called BAliBase (benchmark alignment database) was presented [91]. BAliBase provides more data sets than the McClure tests, in the first version a total of 142 sequence sets were included, sorted into five groups according to the following characteristics: sequences with an overall high (group 1), average (group 2) or low (group 3) similarity, sequence sets with N/C-terminal extensions (group 4) and with internal indels (group 5). This first version of BAliBase has been extended later on in a second version [6] where three additional groups have been added: proteins with repeats (group 6), circular permutations (group 7) and transmembrane proteins (group 8).

The big benefit of this method is the high efforts that have been put in the generation of biologically correct reference alignments: protein structures have been superimposed on the alignments. Furthermore, core regions of the reference were defined. Additionally to the SP scheme comparison with the reference, Thompson *et al.* introduced a *column score* CS. This more sensitive method assigns a score of 1 whenever the complete column of two compared alignments are identical, and 0 otherwise.

$$CS(a_1[i], a_2[i], \ldots, a_k[i]) = \begin{cases} 1 & \text{iff } \exists\, j \text{ such that} \\ & |(\hat{a}_1[j], \hat{a}_2[j], \ldots, \hat{a}_k[j]) = (a_1[i], a_2[i], \ldots, a_k[i]) \\ 0 & \text{otherwise} \end{cases}$$

Although well concepted, BAliBase suffers from serious technical problems in practice. Parts of the provided data sets are incomplete, e.g., reference alignments or annotation files storing the core regions are missing. BAliBase 2 sometimes does not provide global reference alignments at all (group 6). Moreover, the program to compare test alignments with the reference (`baliscore`) is not working correctly. Finally, it has to be stated that BAliBase is not the only source of structure-based alignments. *Homstrad* is a database freely available over the Internet with more structure alignments available but without manually assigned core regions. It was also used already

for benchmarking [95].

2.5.2 Randomized Data Tests

In 2002 a Swedish research group presented a benchmark for multiple alignments based on artificially generated sequences [47]. They used the BAliBase references together with Rose [82], a program simulating evolution of sequences using a probabilistic model. A tree guides the simulated evolution starting from a common ancestor and using operations of the SI model to generate new sequences. As all events in the history of the generated sequences are known, the 'true' alignment of the artificially evolved sequences can be reconstructed. Furthermore, the insertion of more or less random sequences by Rose is stated to compensate for the BAliBase bias for global alignment methods [47]. Applying the protocol as described, many data sets of sequences with different lengths and different evolutionary distances have been generated for alignment benchmarking. A similar method has recently been used to test the ability of alignment methods to align functional noncoding DNA [68].

Lassmann and Sonnhammer also introduced a new score for alignment comparison, the so-called *overlap score*[3]. Here, the number of correctly aligned character tuples is not compared to the number of total tuples in the reference alignment but to the average number of tuples in the reference and in the test alignment. The percentage value expressed by the overlap score therefore also captures the length of the test alignment: for longer alignments with more gaps and therefore less aligned tuples, the average number of aligned tuples decreases and the percent of correctly aligned positions is comparatively high.

[3]Here, the overlap score refers to the overlap of aligned character tuples between reference alignment and test alignment. It must not be confused with the equally named score for overlapping diagonals in the Dialign method (Section 2.4)

2.6 Distance-based Phylogenetic Trees

Phylogenetic analyzes reconstruct the ancestral relationship between *taxa* that in molecular biology often are represented by the sequences of genes or proteins. A simple group of phylogenetic techniques, the so called *distance based methods*, are based on pairwise distance measures $d(s_p, s_q)$ derived from the sequences. Distances can easily be derived from a multiple alignment by counting the mismatching positions. Afterwards, most often a correction for multiple substitutions is applied. These differ in the assumptions made on the distribution of characters from Σ, and popular statistical correction models have been developed by Jukes and Cantor [41], Felsenstein [27] and Kimura [44]. In general, the pairwise distances follow the rules of a metric.

$$\begin{array}{lll} (i) & d(s_p, s_q) > 0 & \forall s_p \neq s_q \\ (ii) & d(s_p, s_q) = 0 & \forall s_p = s_q \\ (iii) & d(s_p, s_q) = d(s_q, s_p) & \forall s_p, s_q \in S \\ (iv) & d(s_p, s_q) \leq d(s_p, s_u) + d(s_u, s_q) & \forall s_p, s_q, s_u \in S \end{array}$$

All pairwise distances are stored in a so called *distance matrix* from which distance based algorithms infer phylogenetic trees. An *additive metrices* has to additionally satisfy the four-point condition of Buneman [16]:

$$d(s_p, s_q) + d(s_u, s_v) \leq d(s_p, s_u) + d(s_q, s_v) = d(s_p, s_v) + d(s_q, s_u).$$

An even harder condition is the three-point condition required for *ultrametric distances*:

$$d(s_p, s_q) \leq d(s_p, s_u) + d(s_u, s_q).$$

Some algorithms (e.g., un/weighted pair group method using arithmetic averages UPGMA/WPGMA and agglomerative clustering) assume ultrametric distances, while others work on roughly additive metrics (e.g., the Neighbor Joining method [72]). Both iteratively join two taxa s_p, s_q with minimal distances into a hypothetical common ancestor u. Without going into detail, the differences are in the calculation of

the distances $d(s_v, u), v \neq p, q$ from the original distances $d(s_v, s_p)$ and $d(s_v, s_q)$. Obviously all methods require $\frac{k(k-1)}{2} = \mathcal{O}(k^2)$ space since quadratically many distance measures are stored. The time complexity of the methods based on ultrametric data is about $\mathcal{O}(k^2 \log k)$, depending on the data structure used to sort the distances after computation. The neighbor joining method additionally comprises a correction step, to force the distances in a more additive metric (i.e., to find a neighbor pair that minimizes the branch-length sum of the resulting tree). The additional step for the rate correction fixes the lower bound for the computation time to $\mathcal{O}(k^3)$ [86, 15].

Chapter 3

Contributions

After the "golden age" of biological sequence analysis in the 1980s and early 1990s, there still remain several unsolved problems. Next to algorithmic refinements, sequence analysis ourdays is also moving away from the classical per-character basis to investigations of functional or structural subunits like genes, domains or repeats. This chapter gives an overview of the contributions to the field of alignment and sequence comparison presented in this thesis. The following papers are included:

Paper I: Michael Sammeth, Jörg Rothgänger, Wolfram Esser, Jürgen Albert, Jens Stoye, and Dag Harmsen (2003) QAlign - Quality Based Alignments with Dynamic Phylogenetic Analysis. *Bioinformatics*, 19, 1592–1593.

Paper II: Michael Sammeth, Burkhard Morgenstern and Jens Stoye (2003) Divide-and-Conquer Alignment with Segment-Based Constraints. Proceedings of ECCB 2003, *Bioinformatics*, 19, II189–II195.

Paper III: Michael Sammeth and Jaap Heringa (2006) Global Multiple Sequence Alignment with Repeats. *Proteins: Structure, Function, and Bioinformatics*, 64, 263–274.

Paper IV: Michael Sammeth and Jens Stoye (2006) Comparing Tandem Repeats with Duplications and Excisions of Variable Degree. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, in press.

Paper IV is a long version of the paper published in the Proceedings of WABI 2005 [76]. Additionally to the papers, related work by others that has not been introduced sufficiently due to space limitations in the publications, extended descriptions of the methodology, developments after publication and possible enhancements of the respective techniques are given.

3.1 Graphical Framework

When looking at the implementation of state of the art multiple alignment algorithms, a tragical observation is inevitable: most of them lack executable files for graphic based platforms such as Windows or Macintosh (e.g., [65, 57]). While this is no problem for using them in automated pipelines running on a UNIX clone operating system, it efficiently prevents these techniques from being used by many experts in the biological lab to find good alignments on sequences to be analyzed.

Focussing on phylogenetic analysis tools, the situation only improves at the first glance. Freely available programs either lack algorithms or graphical capabilities. For instance, the free Phylip package written by Joe Felsenstein [28] is still one of the most elaborated packages. It includes distance-based, Parsimony and Maximum Likelihood algorithms. However, often a chain of programs has to be used in order to perform one phylogenetic analysis. Not rarely the file based interfaces between two modules are making problems, even more if data is imported from an external program. Finally, to plot the data another program is required, since Phylip only outputs trees in the flat file Newick format.

BioEdit is a visual interface for Windows platforms and has been developed at the North Carolina State University. It bundels a series of external programs, e.g., ClustalW [90] and Blast [2] and some programs from the PHYLIP package [28] to calculate distance-based and parsimony trees. BioEdit provides a graphical interface with an alignment viewer with highlighting and editor capabilities. However, since all programs are attached externally via command line interfaces (CLI), the communication with the algorithms is not very flexible, i.e., it does not support all parameters

and results of computation. Although the software provides the possibility to include other programs, these are also limited to the CLI and the problem of getting an executable Windows code of a respective add-on program is delegated to the user. Finally, it is impossible to port the main user interface to another platform since the C source code is not available.

MEGA (molecular evolutionary genetics analysis) is another visual interface for the Microsoft Windows platform. The software also includes the ClustalW alignment program but focusses on the phylogenetic and evolutionary analysis. For the alignment and the original trace files an editor is provided. The phylogenetic module includes many statistical methods to test and to evaluate the tree inference. But also this program is provided *as is* – with only one automatic alignment algorithm and no platform portability, i.e., users of Macintosh are encouraged to use virtual PC emulators.

Here the idea of QAlign was initiated: basically a graphical tool has been created in system-independent JAVA to unite different state-of-the-art techniques for multiple alignment and at the same time incorporate methods for phylogenetic analysis. In order to control the quality of the results, visualizations guide in the comparison. The goals that have been set for the first version are:

- alignment algorithms based on different state of the art methods
- algorithms to infer phylogenetic trees
- an intuitive user interface
- guided editing functionality of multiple alignment layouts
- flexible import and export capabilities for alignments and phylogenetic trees
- multiple platform support, especially versions for Windows and Macintosh systems

Paper I describing the first version of QAlign was published in 2003, and since then it draws the attention of an ever increasing circle of users in microbiology and related fields. Until today nearly 2000 users from all around world have registered for using QAlign, especially in the USA, where subsequent to its publication the journal *The Scientist* included an article about QAlign [39]. The large international resonance has demonstrated the need for such a tool. In a second version we have developed, the concept has been extended towards a general tool for sequence analysis:

- sequence viewers are enhanced to support large sequence sets (e.g., strings on a genomic scale)
- a workspace has been set up to administrate several projects on different sequence sets
- the modular program structure is now easily extensible by plug-in technology

Figure 3.1 and Figure 3.2 show screenshots from the pre-release of QAlign2 *panta rhei*, introduced in June 2005 on the 105th General Meeting of the American Society of Microbiology (ASM) [73]. QAlign2 provides a flexible framework for sequence analysis, primarily multiple alignment and phylogentic reconstruction. Panta rhei includes different algorithms for multiple alignment of which MSA and DCA have been re-implemented in JAVA and ClustalW is attached with a native interface to the original C-code (Figure 3.1). Therefore, both are totally integrated in the graphical framework. T-Coffee and Dialign have been recompiled for the Microsoft Windows and the MacOSX platform, but support many CLI options graphically. The alignments can be viewed, and interesting regions may be highlighted. Moreover, a flexible editing function allows guided editing (Figure 3.1). Distances can be extracted from the alignments, and statistical corrections for multiple substitutions are available, i.e., Jukes-Cantor, Kimura and Felsentsteins model. Different methods for tree inferring (i.e., UPGMA, WPGMA, Agglomerative Clustering and NJ, Figure 3.2) can be used to test and evaluate evolutionary hypotheses. The resulting trees can be browsed,

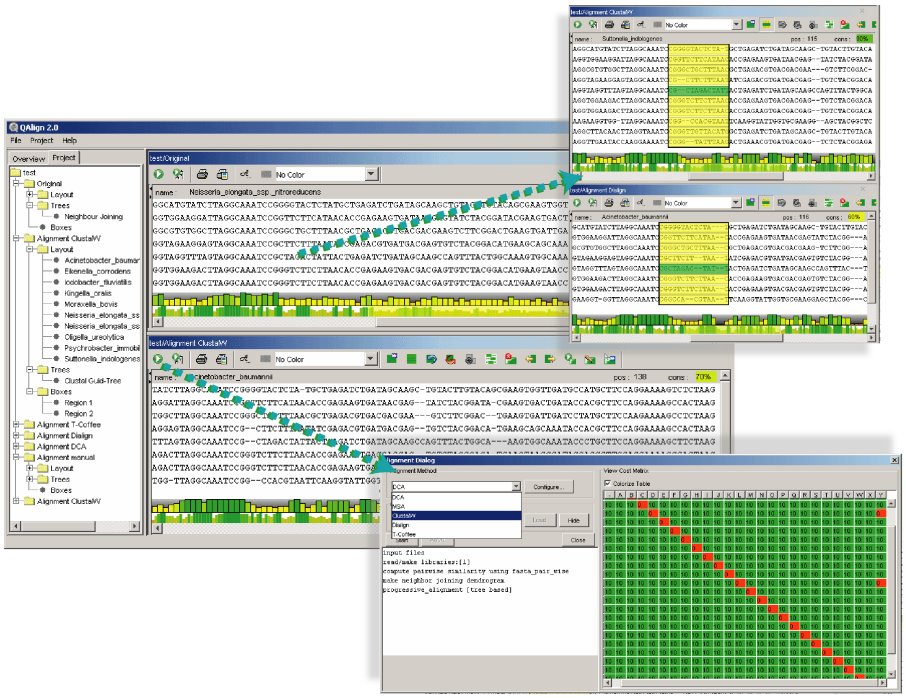


Figure 3.1: The *panta rhei* framework. In the new version of QAlign users can administrate different projects in the workspace (left), each of them may contain various alignments and phylogenetic trees. To generate alignments, a suitable algorithm and corresponding settings may be chosen from a list (bottom-right). A flexible editing functionality allows to move marked blocks within the alignment layout (top-right).

zoomed and nodes may be flipped in order to compare them graphically (Figure 3.2). We are in the process of making the source code publicly available and a second publication on this is currently in preparation.

3.2 Hybrid Algorithms

A well known fact is that the strengths of existing multiple alignment methods differ substantially [92, 47, 68, 95]. This is primarily due to the already mentioned fact that the techniques used are based on different heuristics (Section 2.4). However, also the strategy of looking for similarities (e.g., local vs. global alignment), the objective function and the characteristics of the input data play a role. A recent study

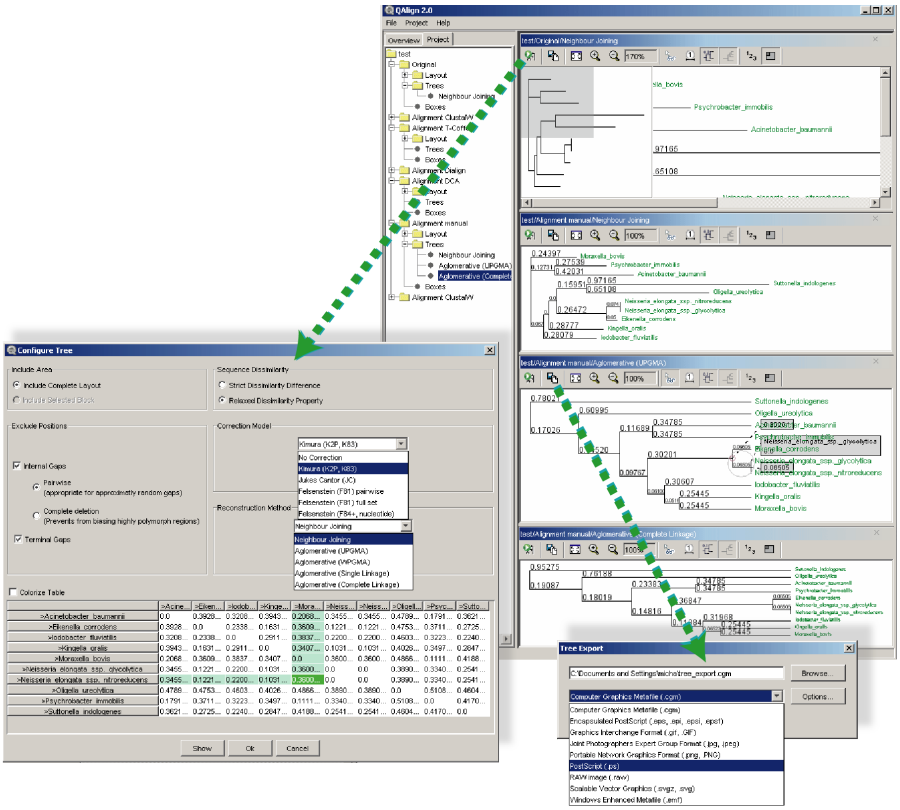


Figure 3.2: To infer trees with QAlign2 the distance matrix can be corrected for multiple substitution before submitting as input to one of the distance-based tree reconstruction methods (bottom-left). Different trees can be generated on the same data set to compare them visually with each other (background). At every time, trees may be exported to a graphic file (bottom-right).

by Lassmann and Sonnhammer [47] demonstrated the effective differences between popular methods like ClustalW [90], Dialign [57], POA [48] and T-Coffee [65]. They performed a series of randomized sequence alignments with the four techniques, artificially varying sequence length and similarity (Section 2.5). Summarizing the results, T-Coffee is more suited for sequence sets with a rather high similarity, while Dialign outperforms the other methods on data sets with low similarity. POA produces overall alignments of good quality, while ClustalW in contrast performs poorer in these tests (Figure 3.3).

These facts make clear that the question for the best alignment algorithm can only

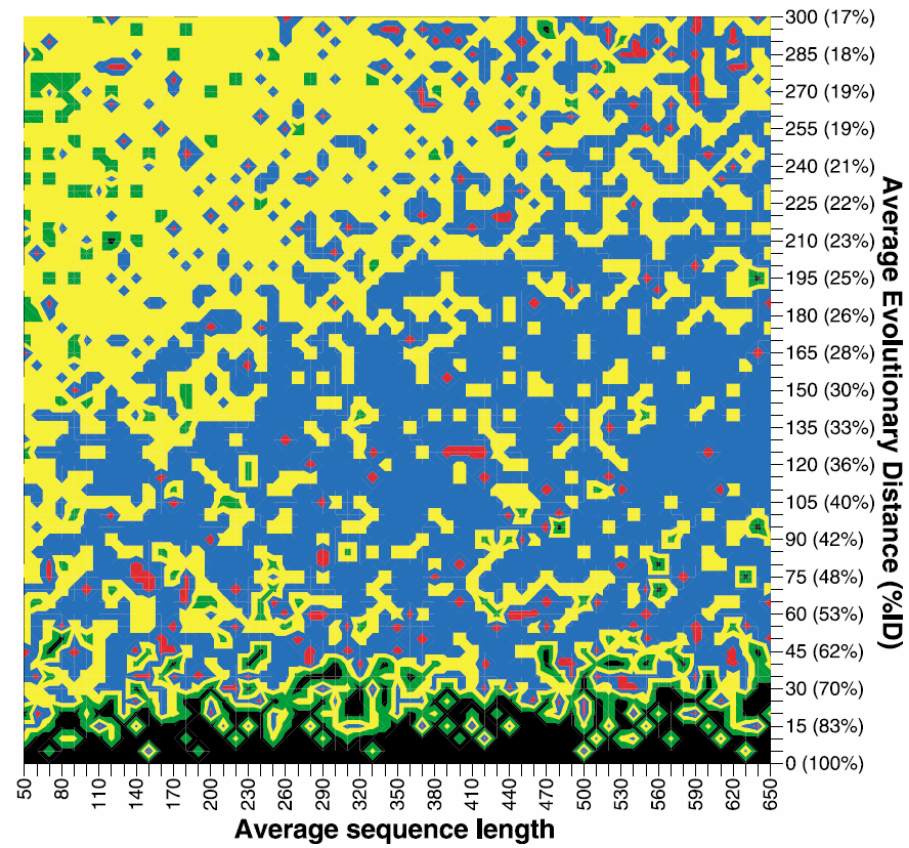


Figure 3.3: Color coded matrix showing which method performed best for each pair-combination of conditions: average sequence length (x-axis) and average evolutionary distance (y-axis). The methods shown are Poa (green), Dialign (yellow), T-Coffee (blue) and ClustalW (red). Picture taken from [47].

be answered if the characteristics of the respective input set are known. However, for the average user a method uniting the strengths of different alignment strategies is highly desirable. Two alignment techniques that are very divergent from the basic strategy are the global divide-and-conquer alignment and Dialign, based on local alignments (Section 2.4). We have developed a *hybrid algorithm* of the two methods where we use anchors found by Dialign to constrain global divide-and-conquer alignment.

The idea of constrained multiple alignments has been investigated before. Myers *et al.* [61] constrained already in 1996 the progressive alignment, yielding an algorithm

with quadratic time effort concerning the number of input sequences as well as their length: $\mathcal{O}(k^2n^2 + k^2|\mathcal{C}|)))$, where k is the number of sequences whose total length is n and $|\mathcal{C}|$ is the number of constraints in the collection $\mathcal{C} = \{C_1, C_2, \ldots, C_{|\mathcal{C}|}\}$.

In contrast to the work of Myers where constraints are set by the user to force certain positions of the input sequences into alignment, the so-called CMSA (constrained multiple sequence alignment) method has been suggested by Tang *et al.* in 2002 [89]. It treats a slightly different problem since the constraints are set on letters from Σ rather than for sequence positions. The biological background for this assumes the presence of a common motif that has to be found in all of the input sequences (e.g., the katalytic triade in RNAses). For a collection of constraints $\mathcal{C} = \{C_1, C_2, \ldots, C_{|\mathcal{C}|}\}$, the original progressive approach by Tang *et al.* used $\mathcal{O}(|\mathcal{C}|kn^4)$ time and $\mathcal{O}(n^4)$ memory [89]. In this first version the constraints only consisted of a single nucleotide or amino acid, $|C_i| := 1$ for all $C_i \in \mathcal{C}$. Here, all constraints $C_i \in \mathcal{C}$ correspond to a single letter. Later, this result was improved to $\mathcal{O}(\alpha k^2 n^2)$ time and $\mathcal{O}(\alpha n^2)$ space, where $\alpha := \sum_{\mathcal{C}}(|C|)$ denotes the lengths sum of all constraints that now can comprise fragments of several nucleotides/amino acids [19]. For some applications, biologists may further expect that some mismatches are allowed among the columns requested to be aligned. Hence, Tsai *et al.* (2004) studied such a kind of constrained sequence alignment and designed an algorithm of same time and space requirement as Chin *et al.* [94]. Finally, Lu *et al.* [94] improved the technique for linear memory requirement $\mathcal{O}(\alpha n)$ by applying techniques developed for the divide-and-conquer alignment [83, 84].

Paper II describes our method and its benefits. In an outline, the algorithm applies the Dialign strategy to greedily collect a set of consistent anchors down to a certain user set threshold T. Anchors with an *olw*-score lower than T are not regarded as significant and thus omitted from the constraints. In a second step, an adapted DCA protocol to accept these anchors is used to align the sequences globally. The modification of DCA comprises two major changes, one in the divide (Theorem 3.2.1) and one in the conquer step (Theorem 3.2.2).

Theorem 3.2.1 (constrained divide step)**.** *If and only if the cut points are consistent with the constraints, substrings can be aligned w.r.t. the sequences.*

Theorem 3.2.1 becomes clear, when imaging as constraint a position tuple of two sequences that are to be aligned with each other. If by the divide step the two positions are separated, they can not be aligned in the conquer step. To avoid such inconsistent cutting, we have to restrict the cut positions to sets that are consistent with the constraints. We use a data structure introduced by Abdeddaïm that stores all constraints [1]. It allows to derive in constant time the so-called *transitivity frontiers*, an upper (lower) bounds $ub(p, i, q)$ ($lb(p, i, q)$) that are indices of the left-most (right-most) characters in sequence s_q that are to the right (left) of i with respect to the constraints.

Algorithm 1 describes how to collect a set of cut positions $\{\hat{c}_2, \ldots, \hat{c}_k\}$ consistent with the constraints for a given cut position $\hat{c}_1$. The algorithm successively "locks" the cut positions $\hat{c}_i$ on all sequences and narrows the according transitivity areas of sequences, where still no cut position has been defined on.

Algorithm 1. *Locking Loop (**sequence** $\{s_1, s_2, \ldots, s_k\}$, **integer** $\hat{c}_1, 1 \leq \hat{c}_1 \leq |s_1|$)*

$lb_1 = ub_1 = \hat{c}_1$
for all $2 \leq p \leq k$ ***do***
  $lb_p = 1$
  $ub_p = |s_p|$
end for
for all $2 \leq p \leq k$ ***do***
  $\hat{c}_p = ub_p - \lfloor \frac{ub_p - lb_p}{2} \rfloor$
  if $ub_p > (lb_p + 1)$ ***then***
    for all $p < q \leq k$ ***do***
      $lb_q = max(lb_q, lb(\hat{c}_p, q))$
      $ub_q = min(ub_q, ub(\hat{c}_p, q))$
    end for
  end if
end for

end

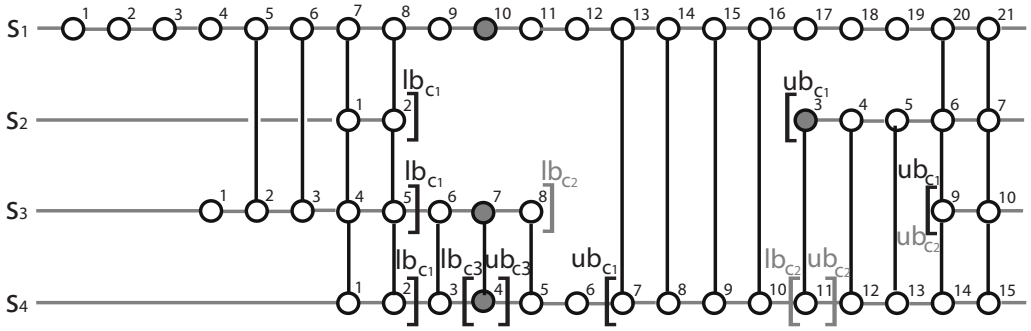


Figure 3.4: Four sequences (horizontal lines) in a constrained dividing step. When finding a set of consistent cut points $\{c_1, c_2, c_3, c_4\}$ (grey positions): a given cut position c_1 on s_1 sets the transitivity frontiers in s_2, s_3 and s_4. In the middle of these bounds iteratively cut points are chosen for the rest of the sequences, which on their turn can further narrow the transitivity interval. Shown are the transitivity frontiers lb_{c_x} and ub_{c_x} induced by every cut point c_x. Note that in case of an empty transitivity interval, arbitrarily the position to the right is taken as a cut point (c_2), but its transitivity frontiers are not respected when narrowing the intervals (lb_{c_2}, ub_{c_2}).

A special case is the selection of a cut position when the transitivity frontiers include an empty interval. Here, one of the adjacent positions has to be chosen as cut point, and we arbitrarily use the position to the right of the interval. However, since this position is not part of the interval, transitivity frontiers implied by it are not respected when narrowing the intervals of other sequences (Figure 3.4).

Additionally, the framework of anchors imposes constraints on the selection of cut points and on the possible alignment paths in the multi-dimensional DP matrix of the conquer step. The proof for Theorem 3.2.2 is straightforward since the final divide-and-conquer alignment is a concatenation of the aligned substrings.

Theorem 3.2.2 (constrained conquer step)**.** *The constraints are respected in the final alignment, if and only if they are respected by all subalignments in the conquer step.*

The method was tested applying the BAliBase (v.1) test sequences. In comparison, the results of DCA, Dialign, ClustalW and T-Coffee are shown. The strengths of different methods are clearly separated, as exclusively global aligners (ClustalW and DCA) create high resoluting alignments where sequence similarity is fairly high (group 1), while algorithms integrating local similarities (Dialign and T-Coffee) are

dominating for input sets with big indels (group 4 and group 5). In all test groups the hybrid method shows characteristics of both underlying techniques, i.e., raising parameter T the attribute of the generated alignment changed from Dialign to DCA (Paper II, Table 1). Also the running time on a certain input set performs according to setting of parameter T (Paper II, Table 2) which we summarize in Theorem 3.2.3. Obviously, although additional time is consumed for the consistency checks, the number of cells omitted in the DP matrix grows with the number of constraints.

Theorem 3.2.3 (speed up)**.** *Constraints speed up the computation of an optimal alignment with the DCA method.*

Selecting the correct T the hybrid method produces results close to the ones yielded with the T-Coffee method, another efficient combination of global and local alignment. However, since the test results are arithmetic averages across a set of very heterogenous data sets, it has to be stressed that in individual instances the hybrid method of Dialign and DCA surely can outperform T-Coffee as demonstrated in Paper II, Table 3.

In future refinements of the method an automatic choice of T according to the attributes of the input would be highly desirable. However, it is quite difficult to find a OF for the automatical analysis of such attributes. Another possibility to circumvent the problem is to change the way the algorithm scores are merged: instead of accepting and discarding anchors by a threshold value, one could think about a weighting scheme similar to the T-Coffee OF [65]. Dialign anchors then positively bias for certain edges in the hyperspace alignment while they penalize other ones, according to the matches of the local alignments. It is only to speculate about the attributes of such a technique, but we expect a quality higher than regular divide-and-conquer, whereas the computation time is likely to be much higher than for the hybrid algorithm presented here.

Another comfortable extension concerns the possibly exponential steps of (i) finding optimal cut points and (ii) aligning the substrings optimally. Although in general both steps are drastically sped up by the constraining anchors found beforehand, no

upper bound for the heuristic can be given since the quality of the anchors relies on similarities of the input sequences. Therefore, there also exist cases where also the hybrid algorithm takes very long to terminate as shown in the publication (Paper II, Table 2). However, both steps (i) and (ii) can be replaced by polynomial time heuristics [81, 35]. These heuristics surely drop the quality of the result, but the speed up also makes the method better comparable in automated benchmarks, e.g. [95].

3.3 Motifs and Repeats

> *Assigned to patriarchal poetry too sue sue sue shall sue sell and magnificent can as coming let the same shall shall shall let it share is share is share shall shall shall shall shell shell shall share is share shell can shell be shell be shell moving in in in inner moving move inner in in inner in meant meant might might may collect collected recollected to refuse what it is is it.* - Patriarchal Poetry, Gertrude Stein

From the poem it should become clear, how repeats confuse us while trying to interprate a sense into the words of the frase. Similar problems arise when aligning proteins containing multiple copies of the same motif (or even of a complete domain), that additionally may be re-ordered and shuffled. *"Repeated sequences (in tandem or not) are renowned for confusing all existing MSA methods."*[64] stated a recent survey. However, repeats cannot be removed from the sequences since the loss of sequence information and of the pattern of repeats omits vital information. Although automated repeat detection [38, 46, 36, 88] has already been investigated for some time, integration of these results with multiple alignment methods has not found much attention until recently [69, 59].

Standard alignment methods (e.g. Dialign and T-Coffee) that are not aware of repeats, often are prune to the strong local similarities between the repeated substrings (Paper III, Table 2). Parts of the repeated motifs are torn apart while aligning them to different repeats of another sequence or they are aligned with random similarities in non-repetetive sequence areas (Paper III, Figure 7 and Figure 8). Although

often the correct alignment has not yet been defined on a certain sequence set, the mentioned misalignments are biologically false, regardless of the mathematical score yielded. Therefore, Definition 3.3.1 is the basis for the measure we use to evaluate our method.

Definition 3.3.1. A *biologically meaningful alignment* of sequences $S = \{s_1, s_2, \ldots, s_k\}$ with repeated motifs $\{f_1, f_2, \ldots, f_l\}$ must not align any position x of $f_v, 1 \leq v \leq l$ with a position y of $f_w, 1 \leq w \leq l, w \neq v$ or a position z that is not part of any motif.

To solve the problem, Raphael *et al.* [69] redefine the representation of a multiple alignment. In contrast to the normal row/column matrix M_S, they visualize alignments as directed, possibly cyclic graphs amongst sequence motifs (so-called *A-Bruijn* graphs). On the one hand this representation efficiently displays all relations within sequence motifs of the input data set, but on the other hand, no preferred alignment is resolved.

More or less at the same time, Morgenstern *et al.* modified the Dialign protocol to provide the input of user-defined constraints (such as motifs or repeats). However, the method is only semi-automated, since the user has to resolve the correct alignment of repeated or shuffled motifs on his own. Moreover, each of the constraints (i.e., the anchors enforced) has to be assigned a weight that is compared to the $olw(\mathcal{A}_{f_x,f_y})$ scores of the anchors found by local alignment when selecting greedily a subset of consistent anchors. But without knowledge about the Dialign OF and the weights of the anchors automatically found in the data set, the assignment of reasonable weights to the user constraints is fairly difficult.

Paper III describes an automated alignment method we have developed that additionally allows to input a set of eventually repeated motifs. This has been realized by a flexible input parameter that models substrings of the input sequences as motifs $f = (b, e, \tau)$. Optionally global alignments of fragments with the same type τ may be submitted, or they are generated automatically by a DCA alignment. Next, our algorithm extracts anchors from these alignments and scores them according to the Dialign OF. Obviously, this requires that the true relationship of repeated motifs is indicated by the character similarity as expressed by Assumption 3.3.1.

Assumption 3.3.1. *In the case of repeats, the most similar motifs are the pair of evolutionarily corresponding ones. They can sometimes be identified by synteny conditions of similarities in flanking substrings.*

Some modifications are to be performed due to limitations of the original Dialign scoring scheme, where an anchor can contain no more than 40 positions, a limitation due to the lack of empiric error probabilities. To this end, we break longer anchors in parts of a maximum length of 40 and multiply the error probabilities, i.e., we add their logarithms for obtaining a weight w. The values obtained approximate the real error probabilities quite good (Morgenstern, pers. communication). Further anchors are added by analyzing separatively those parts of the input sequences not containing fragments. Like in the Dialign procedure, a global alignment is resolved by greedily filtering out a high scoring consistent subset of anchors (Figure 3.5).

In addition, a modification of the algorithm has been developed for noisy signals such as uncertain fragment boundaries (e.g., results of predictions). The extension also comprises a step in which crossreferencing anchors between fragments and non-fragment areas are collected.

In order to evaluate our method, we developed a novel measurement, the separation index SI. It reduces the parameters true/false positives/negatives commonly used for sensitivity and selectivity tests to correctly aligned positions and misalignments, where misaligned positions are biologically wrong according to Definition 3.3.1. To be specific, if both aligned positions come from a designated motif or repeat (of the same type) or both are located in non-motif/non-repeat areas, the alignment is considered biologically correct, otherwise not (misalignment). In benchmarks on the BAliBase repeat group, it could be demonstrated that plain sequence alignment methods (i.e., Dialign and T-Coffee) often misalign positions of motifs with regular substrings and vice versa (Dialign for about 13% of the sequence positions and T-Coffee for about 28%, Table 3.1). In contrast, the motif-constrained method cannot generate such misalignments. The drawback is, however, that erroneous fragment boundaries easily mislead the method. In these cases a slight recovery is still possible when relaxing the constraints (Paper III, Table III).

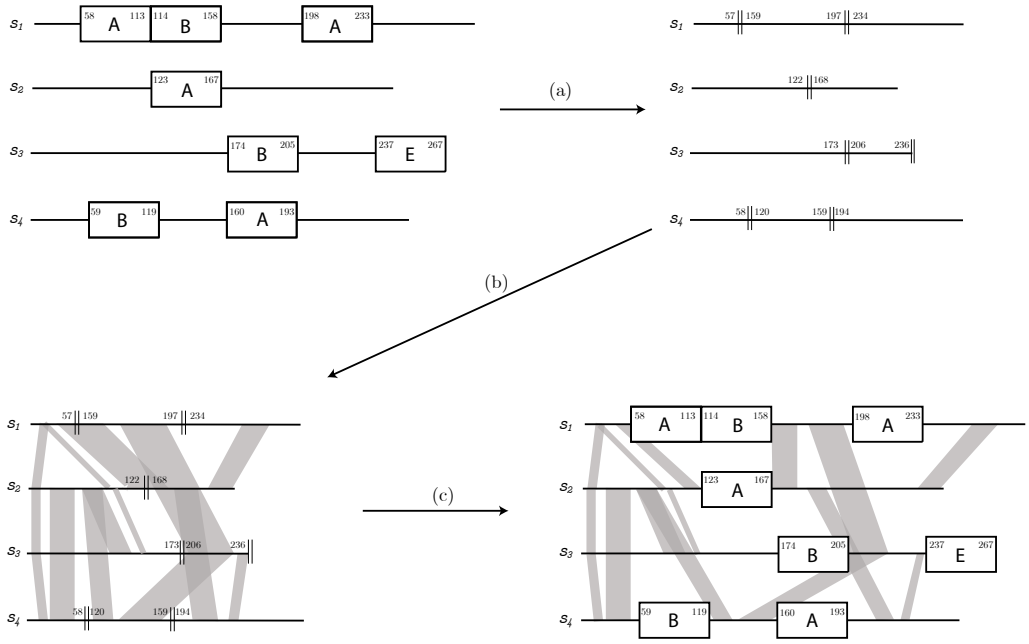


Figure 3.5: Basic outline of finding local similarities in the intervening sequences: (a) Initially, all repeat areas are filtered out, concatenating the intervening sequences at the ‖ marked positions. (b) Next, anchors (grey shadowed links between substrings of sequences) are found by applying the standard Dialign protocol on the non-repeat sequences. (c) Finally, the repeat subsequences are re-inserted and the weight of all anchors found is re-calculated. Some of the grey links (alignments) are broken up upon re-insertion of the motifs.

	$\mathbf{SP}_{repeats}$		$\mathbf{SP}_{xalign}$		SI[%]	
total (99)						
T-Coffee	2,155,984	(2,526,975)	520,622	(703,130)	72.43	(71.50)
Dialign	2,327,404	(2,763,917)	335,598	(429,170)	87.75	(86.96)
RepeatAlign	2,644,668	(3,271,138)	0	(0)	100.00	(100.00)

Table 3.1: Comparison of the novel method (RepeatAlign) with standard alignment methods (T-Coffee, Dialign). In total 99 alignments are comprised in this summary of BAliBase group 6. The correctly aligned repeat tuples (SP_{rep}) and the misaligned tuples (SP_{xaln}) are shown for the core regions and in parentheses for all of the repeat areas (core regions and positions with no structural information). Finally, the separation index SI is given, for the core regions of the repeats and for all the repeat areas in parentheses, respectively.

One possibility for future improvement could be the enhancement for noisy signals. However, preliminary analyzes show that the loss of quality is mostly due to the greedy selection of the anchors. Changing the selection to find optimal sets of fragments would raise the time complexity to an exponential effort [49].

3.4 Spa Typing

Staphylococcus aureus is the predominating cause of nosocomial infections [87], and it is responsible for a wide range of human diseases, including endocarditis, food poisoning, toxic shock syndrome, septicemia, skin infections, soft tissue infections, and bone infections, as well as bovine and ovine mastitis. The recent emergence of community-associated methicillin-resistant *S. aureus* (MRSA) strains [67] further heighten the public health concerns. Thus, understanding the spread of *S. aureus* in hospital in community settings is now of paramount importance.

S. aureus is a heterogenous (polymorphic) species [31] that was recently found to have a clonal population structure [26]. Therefore, it is believed that *S. aureus* does not undergo extensive recombination and diversifies largely by nucleotide mutations. In order to distinguish strains, highly discriminating genetic markers that accumulate rapidly are required. For many years multilocus enzyme electrophoresis and, recently, multilocus sequence typing have been effectively used for this purpose [53, 24]. The latter combines the discriminatory power of seven housekeeping genes encoding surface proteins (*sasA*, *sasB*, *sasD*, *sasE*, *sasF*, *sasH*, and *sasI*).

However, in preliminary work a single technique had been clearly shown to be efficient due to the requirements for rates of accumulating genetic variation: the protein A gene has a rapidly evolving repeat region that is suitable to discriminate species (*spa* typing) [45]. But up to now no automated comparison has been developed for the 24-bp variable-number tandem repeats (VNTR) within the 3' coding region of this gene. The lack of an algorithm mainly stems from the fact that the VNTR evolution also includes complex operations like *tandem duplications* and *excisions* of repeat copies.

Other work in the field of tandem repeats focussed on extending the regular *SI model* (for substitution and indels) to a *DSI model* additionally comprising the evolutionary step of tandem duplication. Benson [10] developed already in 1997 an algorithm to compare sequences under the DSI model. His method is based on a modification of the general DP procedure, called *wraparound dynamic programming*. The wraparound diagonal additionally connects each cell $(i, 1)$ of the matrix with the last cell of the previous row $(i-1, |s_2|)$. Therefore, the sequence of s_2 may be used multiple times for alignment, the essential trick to align duplicated tandem repeats in a DP matrix.

Bérard and Rivals [11] focussed on the comparison of *mini-satellites*, repetitive sequences that are between 10 bp and 100 bp. They formulated an evolutionary model that allows substitution, indels, tandem duplication and tandem deletion and developed a DP recurrence to find the optimal alignment of two minisatellite maps in $\mathcal{O}(n^4)$ time and $\mathcal{O}(n^3)$ space. This effort has been improved by Behzadi to quadratic memory and cubic time [8], that then further has been reduced to cubic complexity on run-length encoded maps [9]. However, the models are constrained to single-copy single-step and cannot easily be extended to multi-copy duplications.

Other work [40, 23, 22, 12, 13] concentrates on finding the minimum cost *tandem repeat history*, a problem firstly described by Fitch [29] that is known to be NP-hard [40, 70]. Elemento and Gascuel [22] have proposed exact algorithms based on the *Most Parsimonious Tree* problem. Without going into detail, heuristics are set up by either restricting the problem [40] or the search for an optimal history tree [23, 12, 13]. But all these works do not consider the algorithmical advantages of a second repeat sequence that is to be compared to.

Consequently, in Paper IV we have described the *EDSI model* that in addition to point mutations and tandem duplications also allows the excision of one or more repeat copies (multi-copy, single-step). The duplications and excisions are fixed to the repeat boundaries such that we can replace the nucleotide sequence of the input by a sequence over the macro alphabet Σ' of the tandem repeats (Section 1.4.3). In

case of *S. aureus* the already established *Kreiswirth notation* [45] to identify repeats is used. Once an input sequence s has been transformed to a string s' of letters from Σ', we define the costs of the four edit operations of the EDSI model. In order to get a well-defined computational problem, all costs are obliged to fulfill the requirements of a metric and excision costs have to be greater than zero: with zero cost excisions one easily could create an identity relation between two arbitrary sequences by concatenating both into a common ancestor string from which each of the two sequences can be derived by an excision event.

Mutation (including substitutions and indels): costs between different repeats, i.e., different characters of the macro alphabet, are assigned according to a global alignment of all repeats.

Duplication: the costs for producing a exact copy of a substring. Note that costs for mutation events that change the one or the other repeat copy after duplication can be additionally imposed.

Excision: the costs of deleting one or more repeat copies from the sequence. According to biological observations these costs are generally independent of the length of the deleted string.

Using the EDSI model of evolution, we set up an exhaustive algorithm to compare all possible ancestor strings of two given VNTR clusters. The technique uses an high-dimensional DP procedure that is taking into account also submatrices spanned by possible ancestors. A main result of this paper is a theorem proving the finiteness of the search space in our formulation of the EDSI alignment problem: Theorem 3.4.1 (finiteness; Theorem 1 in Paper IV). The excisions under EDSI allow to theoretically generate an unlimited number of possible ancestors for a certain sequence. To be specific, between each two adjacent repeats (and at the beginning and the end of the sequence respectively) an unlimited number of repeats can be inserted which hypothetically have been removed by an excision event. However, since our algorithm aims at finding a minimum distance for a compared sequence pair, the number of

ancestors to be regarded can be limited to not redundant ancestors. The latter are ancestors where a shorter distance between the sequences is given by another duplication history, e.g. ancestors with inserted substrings that are related to neither of the compared sequences.

Theorem 3.4.1 (finiteness)**.** *The edit operations under EDSI evolution and their unrestricted order basically force us to explore an infinite search space of possible ancestor sequences. However, the space of operation sets to be explored in order to find the minimal distance between two sequences $d(s', t')$ is finite.*

Our method is based on *contramers*, geometrical constructs that capture a duplication with possible mutations. They are defined by $C = (s', b, m, e, \mathcal{A})$ and represent a contraction, the inverse operation of a duplication, of the two substrings $s'[b, (m-1)]$ and $s'[m, e]$. Corresponding positions are described by the alignment $\mathcal{A}$ of these substrings. Each contramer is also assigned a cost, for the duplication operation and mutations captured by $\mathcal{A}$. In a first step we exhaustively collect all contramers on s' and t', the two sequences to be compared (*primary library*). Then, these contramers are merged to yield cascaded duplication events until all possible tandem repeat histories are generated (*secondary library*). In a high-dimensional alignment, s' and t' are aligned regarding their possible repeat histories that give the minimum distance between the sequence pair (Figure 3.6).

A difficult part is the assignment of the minimum costs for merged contramers. As pointed out, we assume commutativity on the order of merging overlapping duplication events (Theorem 3 of Paper IV). Moreover, there are multiple ways for the transitive relation of repeats joined by a deletion (Paper IV, Figure 4) and we take the possibility producing the least cost when comparing with the other sequence. To do so, we apply a version of the exact MSA algorithm, where the cost function was modified to meet the requirements of the contramers. The difficulty lies in the fact that the ancestor repeat and the time a certain mutation did happen, i.e. before or after duplication, is not given by the contramers. Hence, we try exaustively all positions of a column as ancestor and compare the mutation costs w.r.t. the compared sequence (Figure 3.7).

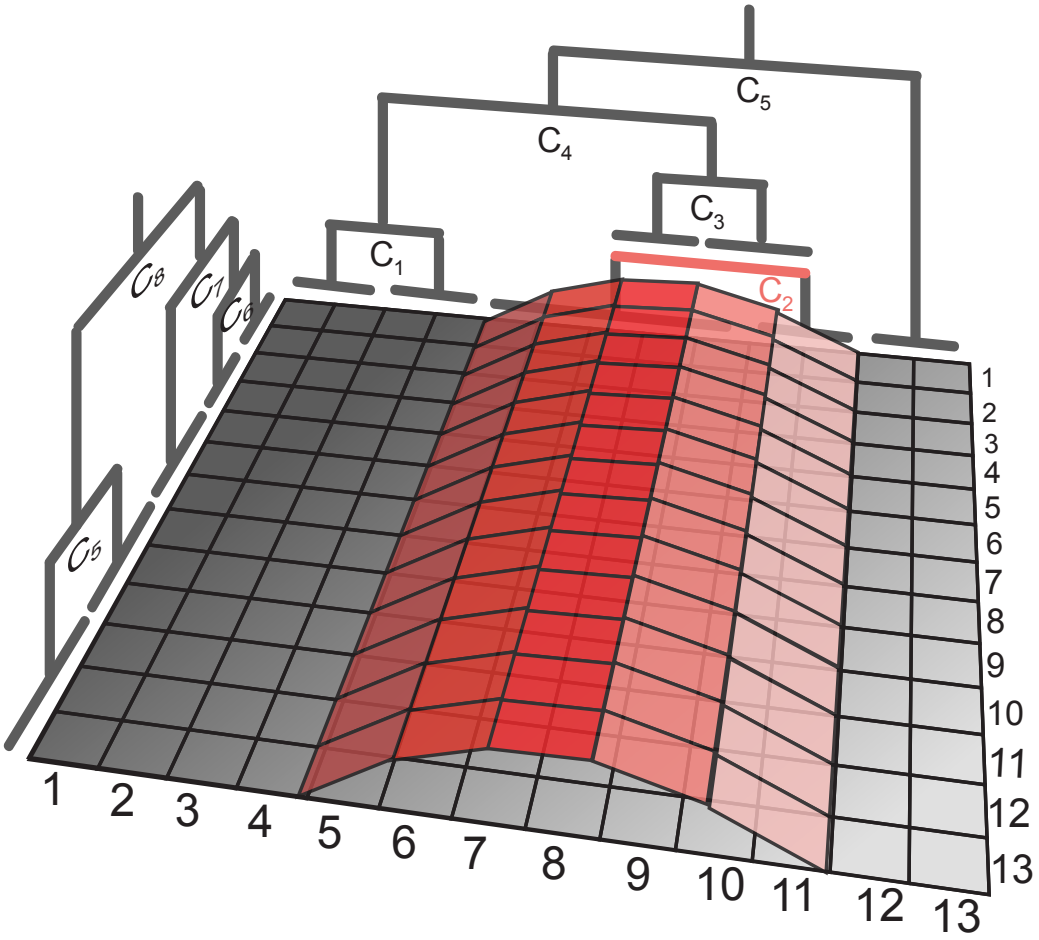


Figure 3.6: An example for an alternative submatrix within a DP matrix $M_{s',t'}$. During the DP process paths within the original and within the submatrix are taken into account when determining the optimum of the cells in column 12. Note that only contramers of one possible repeat history are depicted here, but all cascaded duplication events of the secondary library are investigated.

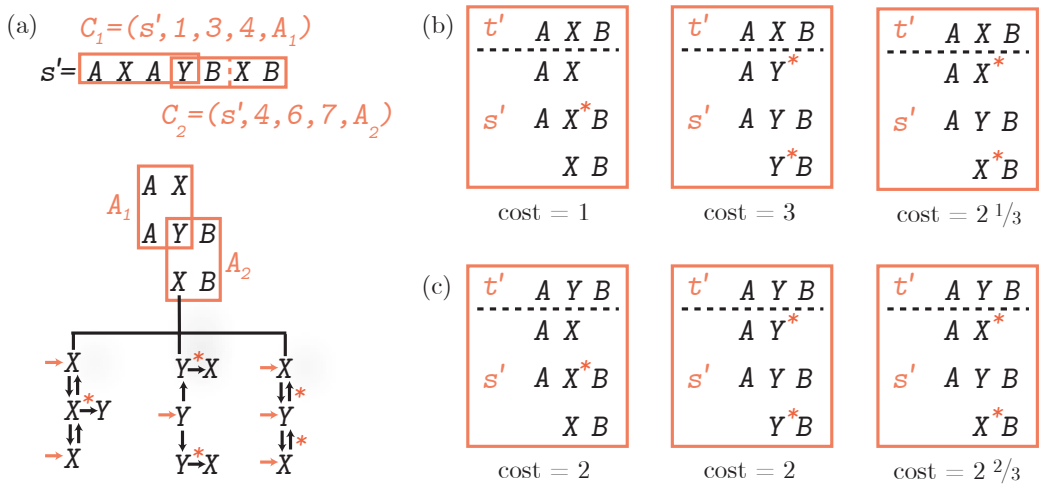


Figure 3.7: Depicted is the scoring function on one ambiguous column of two merged contramers. (a) Since the true ancestor repeat is unknown, all non-identical adjacent positions are considered as possible ancestors (arrows) and the number of mutations (asterisk) is counted when changing the respective position before and after duplication. Note that for completely overlapping contramers, more combinations are possible, since jumps are allowed between the rows (Paper IV, Figure 9c). (b) and (c) demonstrate how the costs also depend on the compared sequence string. All three possible histories of (a) are shown when comparing against the respective target sequence t'.

The time and memory efforts of the method are exponential, due to the size of the exhaustively treated *Tandem History Problem* [40]. Future developments will focus on speeding up the computation, e.g. by introducing a bounding step in the search for contramers. Specifically, we want to exclude the computation of contramers where the contraction and mutation costs do exceed the costs of an already found alignment. Obviously, such a branch-and-bound version will still retrieve the minimum distance.

Chapter 4

Conclusion and Future Aspects

The results of this thesis successfully have tackled a series of severe problems for multiple sequence alignment. One main part of the thesis has investigated the possibilities of *multiple alignment with constraints*. These alignments are taking into account restrictions on the alignment of certain positions of the input sequences. In a first step, we have constrained the global divide-and-conquer alignment to respect as constraints high-scoring anchors found before by Dialign. We have demonstrated that the constrained (hybrid) alignment shares attributes of both underlying techniques. In other words, anchors from local alignments have efficiently been integrated into a global alignment procedure making it more sensitive to weak local signals in the sequence set. Additionally, we have shown how anchors reduce the computation of hyperspace alignments (Section 3.2 and Paper II).

In an extension, we have modified the algorithm to be suitable for aligning sequences containing *repeats*. Here, we have used extended information of repeated sequence motifs to cluster the positions from the input sequences into two groups, repeats and intervening sequence areas. According to the respective stringency mode, the algorithm allows cross-relations or separately investigates both groups for anchors. The anchors then are weighted using the Dialign OF, but the overlapping weight function is changed to not bias the repeat signal (Section 3.3). Finally, the constrained divide-and-conquer alignment reconstructs a global alignment with the

repeat-sensitive anchors as constraints. Paper III shows how we constrain the alignment for proteins with multiple, eventually repeating, motifs and match corresponding motifs to produce biologically more correct results.

Constrained alignment methods are grow more and more important. The number of sequences in databases where additional information to the bare sequence is available, grows nearly every day, e.g. protein family sets (Pfam [7]) and genomes with predicted or manually annotated gene borders (Ensembl [14]). In the future one could think of a wider spectrum for the constraints, e.g., superimposed structural elements and pre-aligned motifs on proteins. In these terms, an interesting point is the application of the technique to DNA. Probably it could enhance the alignment of genes, when the structure of exons (corresponding to the motifs) and introns is known. With modifications it will be possible to use the OF also on a genomic scale where it could be used to gain hints on orthologous (aligned) and paralogous (unaligned) genes. Obviously, it may happen that information overlaps and signals from different sources have to be added. In the so-called post-genomic area where more and more genomic sequences become freely available with predicted or manually annotated information. Surely the constrained alignment method can flexibly be used to integrate expert knowledge into the alignment construction process.

However, in some instances repeats are not disturbing but adding valuable information for sequence comparison. As demonstrated for *spa* typing, the discriminatory power of alignments can generate sensitive distances to create new distance-based evolutionary measures (Section 3.4 and Paper IV). Here a *high-dimensional alignment* has been developed to take into account alignment alternatives (repeat histories) when determining the minimum distance (i.e., the optimal alignment) between a compared sequence pair. These measures surely can be used to infer phylogenetic relations, such as trees. In future efforts we will biologically analyze the resolution ability of the 3'-VNTR cluster of *S. aureus* by the distances produced as described in Paper IV. However, regarding the already mentioned availability of genomic data, it would also be interesting to apply our method to other mini- (19–40 bp) or micro-satellite

(2–6 bp) sets. These tandem repeats occuring in higher organisms also carry a sensitive phylogenetic signal, and were already used to investigate the Out-Of-Africa hypothesis of human origin [4].

Finally, a chapter becoming more and more important for reasonable sequence analysis is the visualization. On the one hand the ever growing data masses can hardly be overviewed, on the other hand also more compact data sets often require special visualization features that highlight sequence characteristics such as repeats. We have introduced QAlign, a flexible and convenient graphical framework for sequence analysis (Section 3.1) and provide it freely on the internet. It is designed to integrate tools for all areas of sequence analysis. QAlign especially focusses on multiple alignment and phylogenetic reconstruction, including characteristic evaluation and context or relation visualization. The second version *panta rhei* provides many plug-in interfaces and viewers for long genomic sequences. Future efforts have to be spent to include genome browser capabilities and other visualizations, e.g. alternative alignment representations like the ABA-graphs introduced by [69].

Bibliography

[1] S. Abdeddaïm, *Incremental computation of transitive closure and greedy alignment*, Proc. CPM 1997, LNCS, vol. 1264, 1997, pp. 167–179.

[2] F.W. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, *Basic local alignment search tool*, J. Mol. Biol. **215** (1990), 403–410.

[3] S.F. Altschul, *Gap costs for multiple sequence alignment*, J. theor. Biol. **138** (1989), 297–309.

[4] J.A. Armour, T. Anttinen, C.A. May, E.E. Vega, A. Sajantila, J.R. Kidd, K.K. Kidd, J. Bertranpetit, S. Pääbo, and A.J. Jeffreys, *Minisatellite diversity supports a recent african origin for modern humans*, Nat. Genetics **13** (1996), 154–160.

[5] O.T. Avery, C.M. MacLeod, and M. McCarty, *Studies on the chemical nature of the substance inducing transformation of pneumococcal types: induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type iii*, J. Exp. Med. **79** (1944), 137–158.

[6] A. Bahr, J.D. Thompson, J.-C. Thierry, and O. Poch, *BAliBase (benchmark alignment database): enhancements for repeats, transmembrane sequences and circular permutations*, Nucleic Acids Research **29** (2001), 323–326.

[7] A. Bateman, L. Coin, R. Durbin, R.D. Finn, V. Hollich, S. Griffiths-Jones, A. Khanna, M. Marshall, S. Moxon, E.L.L. Sonnhammer, D.J. Studholme,

C. Yeats, and S.R. Eddy, *The pfam protein families database*, Nucl. Acids Res. **32** (2004), D138–D141.

[8] B. Behzadi and J.-M. Steyaert, *An improved algorithm for the generalized comparison of minisatellites*, Proc. CPM 2003, LNCS, vol. 2676, 2003, pp. 32–41.

[9] B. Behzadi and J.-M. Steyaert, *The minisatellite transformational problem revisted*, Proc. WABI 2004, LNCS, vol. 3240, 2004, pp. 310–320.

[10] G. Benson, *Sequence alignment with tandem duplication*, Journal of Comput. Biol. **4** (1997), 351–367.

[11] S. Bérard and E. Rivals, *Comparison of minisatellites*, RECOMB 2002 (2002), 67–76.

[12] D. Bertrand and O. Gascuel, *Topological rearrangements and local search method for tandem duplication trees*, WABI (2004), 374–387.

[13] D. Bertrand and O. Gascuel, *Topological rearrangements and local search method for tandem duplication trees*, Transactions on computational biology and bioinformatics **2** (2005), 1–13.

[14] E. Birney, T.D. Andrews, P. Bevan, M. Caccamo, Y. Chen, L. Clarke, G. Coates, J. Cuff, V. Curwen, T. Cutts, T. Down, E. Eyras, X.M. Fernandez-Suarez, P. Gane, and B. *et al.* Gibbins, *Overview of ensembl*, Genome Res. **14** (2004), 925–928.

[15] G.S. Brodal, R. Fagerberg, T. Mailund, C.N. Pedersen, and D. Phillips, *Speeding up neighbour-joining tree construction*, Technical Report (2003), ALCOMFT–TR–03–102.

[16] P. Buneman, *The recovery of trees from measures of dissimilarity*, Mathematics and the archeological and historical sciences (J.H. *et al.*, ed.), Edinburgh University Press, 1971.

[17] J.H. Camin and R.R. Sokal, *A method for deducing branching sequences in phylogeny*, Evolution **19** (1965), 311–326.

[18] H. Carrillo and D. Lipman, *The multiple sequence alignment problem in biology*, SIAM J. Applied Math. **48** (1988), 1073–1082.

[19] F.Y.L. Chin, N.L. Ho, T.W. Lam, P.W.H. Wong, and M.Y. Chan, *Efficient constrained multiple sequence alignment with performance guarantee*, Proc. CSB'03, IEEE, 2003, pp. 337–346.

[20] V.E. Coyne, M.D. James, S.J. Reid, and E.P. Rybicki, *Standard pcr protocol*, Molecular Biology Techniques Manual, `http://www.mcb.uct.ac.za/manual/MolBiolManual.htm`, 1994.

[21] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt, *A model of evolutionary change in proteins*, Atlas Prot. Seq. Struct. **5** (1978), 345–352.

[22] O. Elemento and O. Gascuel, *An efficient and accurate distance based algorithm to reconstruct tandem duplication trees*, Bioinformatics **18** (2002), S92–S99.

[23] O. Elemento, O. Gascuel, and M.-P. Lefranc, *Reconstructing the duplication history of tandemly repeated genes*, Mol. Biol. Evol. **19** (2002), 278–288.

[24] M.C. Enright, N.P. Day, C.E. Davies, S.J. Peacock, and B.G. Spratt, *Multilocus sequence typing for characterization of methicillin-resistant and methicillin-susceptible clones of* Staphylococcus aureus, J. Clin. Microbiol. (2000), 1008–1015.

[25] X. Estivill, M. Chillon, T. Casals, A. Bosch, N. Morral, V. Nunes, P. Gasparini, A. Seia, P.F. Pignatti, G. Novelli, and et. al., *Delta f508 gene deletion in cystic fibrosis in southern europe*, Lancet (1989), 1404.

[26] E.J. Feil, J.E. Cooper, H. Grundmann, D.A. Robinson, M.C. Enright, T. Berendt, S.J. Peacock, J.M. Smith, M. Murphy, B.G. Spratt, C.E. Moore,

and N.P. Day, *How clonal is* Staphylococcus aureus*?*, J. Bacteriol. **185** (2003), 3307–3316.

[27] J. Felsenstein, *Evolutionary trees from dna sequences: a maximum likelihood approach*, J. Mol. Evol. **17** (1981), 368–376.

[28] J. Felsenstein, *PHYLIP (phylogeny interface package)*, Department of Genetics, University of Washington, Seattle, 1993.

[29] W. Fitch, *Phylogenies constrained by cross-over process as illustrated by human hemoglobins in a thirteen-cycle, eleven amino-acid repeat in human apolipoprotein a-i*, Genetics **86** (1977), 623–644.

[30] W.M. Fitch and E. Margoliash, *Construction of phylogenetic trees*, Science **155** (1967), 279–284.

[31] J.R. Fitzgerald, D.E. Sturdevant, S.M. Mackie, S.R. Gill, and J.M. Musser, *Evolutionary genomics of* Staphylococcus aureus*: insights into the origin of methicillin-resistant strains and the toxic shock syndrome epidemic*, Proc. Natl. Acad. Sci. USA, vol. 98, 2001, pp. 8821–8826.

[32] M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of np-completeness*, W.H. Freeman and Company, New York-San Francisco, 1979.

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

[34] O. Gotoh, *Multiple sequence alignment: algorithms and applications*, Adv. Biophys. **39** (1999), 159–206.

[35] S. Gupta, J. Kececioglu, and A. Schäffer, *Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment*, J. Comp. Biol. **2** (1995), 459–472.

[36] J. Heger and L. Holm, *A method to recognise distant repeats in protein sequences*, Proteins: Structure, Function and Genetics **17** (1993), 391–411.

[37] S. Henikoff and J.G. Henikoff, *Amino acid substitution matrices from protein blocks*, PNAS **89** (1992), 10915–10919.

[38] A. Heringa and P. Argos, *Rapid automatic detection and alignment of repeats in protein sequences*, Proteins **41** (2000), 224–237.

[39] S. Jaffe, *Putting a pretty face on multiple alignment*, The Scientist (2003).

[40] D. Jaitly, P.K., G.-H. Lin, and B. Ma, *Reconstructing the duplication history of tandemly repeated genes*, Journal of Computer and System Sciences **65** (2002), 494–507.

[41] T.H. Jukes and C.R. Cantor, *Evolution of protein molecules*, Academic Press, New York, 1969.

[42] J. Kececioglu, *The maximum weight trace problem in multiple sequence alignment*, Proc. CPM 1993, LNCS, vol. 684, 1993, pp. 106–119.

[43] H.G. Khorana, H. Buchi, H. Ghosh, N. Gupta, T.M. Jacob, H. Kossel, R. Morgan, S.A. Narang, E. Ohtsuka, and R.D. Wells, *Polynucleotide synthesis and the genetic code*, Cold Spring Harb. Symp. Quant. Biol. **31** (1966), 39–49.

[44] M. Kimura and T. Ohta, *On the stochastic model for estimation of mutational distance between homologous proteins*, J. Mol. Evol. **2** (1972), 87–90.

[45] L. Koreen, S.V. Ramaswamy, E.A. Graviss, S. Naidich, J.M. Musser, and B.N. Kreiswirth, *Spa typing method for discriminating among staphylococcus aureus isolates: implications for use of a single marker to detect genetic micro- and macrovariation*, J Clin. Microbiol. **42** (2004), 792–799.

[46] S. Kurtz and C. Schleiermacher, *REPuter: fast computation of maximal repeats in complete genomes*, Bioinformatics **15** (1999), 426–427.

[47] T. Lassmann and E. Sonnhammer, *Quality assessment of multiple alignment programs*, FEBS Lett. **529** (2002), 126–130.

[48] C. Lee, C. Grasso, and M.F. Sharlow, *Multiple sequence alignment using partial order graphs*, Am. J. Infect. Control **18** (2002), 452–464.

[49] H.-P. Lenhof, B. Morgenstern, and K. Reinert, *An exact solution for the segment-to-segment multiple sequence alignment problem*, Proc. GCB'98, 1998.

[50] M. Lermen and K. Reinert, *The practical use of the A*-algorithm for exact multiple sequence alignment*, J. Comp. Biol. **7** (1997), 655–671.

[51] B. Lewin, *Genes iv*, Oxford Univ. Pr., 1997.

[52] B. Maddox, *Rosalind franklin the dark lady of dna*, HarperCollins, New York, 2002.

[53] M.C. Maiden, J.A. Bygraves, E. Feil, G. Morelli, J.E. Russell, R. Urwin, Q. Zhang, J. Zhou, K. Zurth, D.A. Caugant, I.M. Feavers, M. Achtman, and B.G. Spratt, *Multilocus sequence typing: a portable approach to the identification of clones within populations of pathogenic microorganisms*, Proc. Natl. Acad. Sci. USA, vol. 95, 1998, pp. 3140–3145.

[54] A.M. Maxam and W. Gilbert, *A new method for sequencing dna*, Proc. Natl. Acad. Sci. USA, vol. 74, 1977, pp. 560–564.

[55] M.A. McClure, T.K. Vasi, and W.M. Fitch, *Comparative analysis of multiple protein-sequence alignment methods*, Mol. Biol. Evol. **11** (1994), 571–592.

[56] F. Miescher, *Die histochemischen und physiologischen arbeiten, gesammelt und herausgegeben von seinen freunden*, Verlag von F.C.W. Vogel, Leipzig, 1897.

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

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

[59] B. Morgenstern, S.J. Prohaska, N. Werner, J. Weyer-Menkhoff, I. Schneider, A.R. Subramanian, and P.F. Stadler, *Multiple sequence alignment with user-defined constraints*, German Conference on Bioinformatics, Lecture Notes in Informatics, 2004.

[60] K.B. Mullis, *The unusual origin of the polymerase chain reaction*, Scientific American **262** (1990), 56–65.

[61] G. Myers, S. Selznick, Z. Zhang, and W. Miller, *Progressive multiple alignment with constraints*, J. Comp. Biol. **3** (1996), 563–572.

[62] S.B. Needleman and C.D. Wunsch, *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, J. Mol. Biol. **48** (1970), 443–453.

[63] M.W. Nirenberg and H.J. Matthaei, *The dependence of cell-free protein synthesis in e. coli upon naturally occurring or synthetic polyribonucleotides*, Proc. Natl. Acad. Sci. USA, vol. 47, 1961, p. 1589.

[64] C. Notredame, *Recent progress in multiple sequence alignment: a survey*, Pharmacogenomics **3** (2002), 131–144.

[65] C. Notredame, D. Higgins, and J. Heringa, *T-COFFEE: a novel method for fast and accurate multiple sequence alignment*, J. Mol. Biol. **302** (2000), 205–217.

[66] P. Nyrén and A. Lundin, *Enzymatic method for continous monitoring of inorganic pyrophosphatase synthesis*, Anal. Biochem. **151** (1985), 504–509.

[67] K. Okuma, K. Iwakawa, J.D. Turnidge, W.B. Grubb, J.M. Bell, F.G. O'Brien, G.W. Coombs, J.W. Pearman, F.C. Tenover, M. Kapi, C.Tiensasitorn, T. Ito,

and K. Hiramatsu, *Dissemination of new methicillin-resistant* Staphylococcus aureus*: identification of two ancestral genetic backgrounds and the associated mec elements*, Microb. Drug Resist. **7** (2002), 349–361.

[68] D.A. Pollard, C.M. Bergman, J. Stoye, S.E. Celniker, and M.B. Eisen, *Benchmarking tools for the alignment of functional noncoding dna*, BMC Bioinformatics **5** (2004), `http://www.biomedcentral.com/1471--2105/5/6`.

[69] B. Raphael, D. Zhi, H. Tang, and P. Pevzner, *A novel method for multiple alignment of sequences with repeated and shuffled elements*, Genome Research **14** (2004), 2336–2346.

[70] E. Rivals, *A survey on algorithmic aspects of tandem repeats evolution*, International Journal of Foundations of Computer Science **15** (2004), 225–257.

[71] R.K. Saiki, S. Scharf, F. Faloona, K.B. Mullis, G.T. Horn, H.A. Ehrlich, and N. Arnheim, *Enzymatic amplification of beta-globin genomic sequences and restriction site analysis for diagnosis of sickle cell anemia*, Science **230** (1985), 1350–1354.

[72] N. Saitou and M. Nei, *The neighbor-joining method: a new method for reconstructing phylogenetic trees*, Mol. Biol. Evol. **4** (1987), 406–425.

[73] M. Sammeth, J.T. Griebel, F. Tille, and J. Stoye, *Qalign2: Panta rhei*, 105th General Meeting of the American Society of Microbiology, 2005.

[74] M. Sammeth, B. Morgenstern, and J. Stoye, *Divide-and-conquer alignment with segment-based constraints*, Bioinformatics **19** (2003), ii189–ii195.

[75] M. Sammeth, J. Rothgänger, W. Esser, J. Albert, J. Stoye, and D. Harmsen, *Qalign - quality based alignments with dynamic phylogenetic analysis*, Bioinformatics **19** (2003), 1592–1593.

[76] M. Sammeth, T. Weniger, D. Harmsen, and J. Stoye, *Alignment of tandem repeats with excision, duplication, substitution and indels (edsi)*, LNBI (Proceedings of WABI 2005) **3692** (2005), 276–290.

[77] F. Sanger, S. Nicklen, and A.R. Coulson, *Dna sequencing with chain-terminating inhibitors*, Proc. Natl. Acad. Sci. USA, vol. 74, 1977, pp. 5463–5467.

[78] P.H. Sellers, *On the theory and computation of evolutionary distances*, SIAM J. Appl. Math. **26** (1974), 787–793.

[79] B. Shopsin, M. Gomez, O. Montgomery, D.H. Smith, M. Waddington, D.E. Dodge, D.A. Bost, M. Riehman, S. Naidich, and B.N. Kreiswirth, *Evaluation of protein a gene polymorphic region dna sequencing for typing of staphylococcus aureus strains*, J. Clin. Microbiol. **37** (1999), 3556–3563.

[80] W.B. Stanley, *Isolation of a crystalline protein possessing the properties of tobacco-mosaic virus*, Science **81** (1935), 644–645.

[81] J. Stoye, *Multiple sequence alignment with the divide-and-conquer method*, Gene **211** (1998), GC45–GC56.

[82] J. Stoye, D. Evers, and F. Meyer, *Rose: generating sequence families*, Bioinformatics **14** (1998), 157–163.

[83] J. Stoye, V. Moulton, and A.W.M. Dress, *DCA: an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment*, Comput. Appl. Biosci. **13** (1997), 625–626.

[84] J. Stoye, S.W. Perrey, and A.W.M. Dress, *Improving the divide-and-conquer approach to sum-of-pairs multiple sequence alignment*, Appl. Math. Lett. **10** (1997), 67–73.

[85] L. Stryer, *Biochemie*, Spektrum Akademischer Verlag, 1996.

[86] J.A. Studier and K.J. Keppler, *A note on the neighbor-joining method of saitou and nei*, Mol. Biol. Evol. **5** (1988), 729–731.

[87] National Nosocomial Infections Surveillance System, *National nosocomial infections surveillance (nnis) system report, data summary from january 1990 – may 1999, issued june 1999*, Am. J. Infect. Control **27** (1999), 520–532.

[88] R. Szklarczyk and J. Heringa, *Tracking repeats using significance and transitivity*, Bioinformatics, Suppl. 1 **20** (2004), I311–I317.

[89] C.Y. Tang, C.L. Lu, M.D.-T. Chang, Y.-T. Tsai, Y.-J. Sun, K.-M. Chao, J.-M. Chang, Y.-H. Chiou, C.-M. Wu, H.-T. Chang, and W.-I. Chou, *Constrained multiple sequence alignment tool development and its application to rnase family alignment*, Proc. 1st IEEE CSB Conference, 2002, pp. 127–137.

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

[91] J. Thompson, F. Plewniak, and O. Poch, *BAliBase: a benchmark alignment database for the evaluation of multiple alignment programs*, Bioinformatics **15** (1999), 87–88.

[92] J. Thompson, F. Plewniak, and O. Poch, *A comprehensive comparison of multiple sequence alignment programs*, Nucl. Acids Res. **27** (1999), 2682–2690.

[93] U. Tönges, S. Perrey, J. Stoye, and A. Dress, *A general method for fast multiple sequence alignment*, Gene **172** (1996), GC33–GC41.

[94] Y.-T. Tsai, Y.P. Huang, C.T. Yu, and C.L. Lu, *Music: a tool for multiple sequence alignment with constraints*, Bioinformatics **20** (2004), 2309–2311.

[95] I.M. Wallace, O. O'Sullivan, and D.G. Higgins, *Evaluation of iterative alignment algorithms for multiple alignment*, Bioinformatics **21** (2005), 1408–1414.

[96] L. Wang and T. Jiang, *On the complexity of multiple sequence alignment*, J. Comput. Biol. **1** (1994), 337–348.

[97] M.S. Waterman, *Sequence alignments*, Mathematical methods for DNA sequences, CRC Press, Boca Raton, Florida, 1989.

[98] M.S. Waterman, *Introduction to computational biology: Maps, sequences, and genomes*, CRC Press, 1995.

[99] M.S. Waterman, T.F. Smith, and W.A. Beyer, *Some biological sequence metrics*, Adv. Math. **20** (1976), 367–387.

[100] J.D. Watson and F.H.C. Crick, *Molecular structure of nucleic acids*, Nature **171** (1953), 737–738.

Acronyms

CLI: command line interface. The program and parameters a program call provides from the command line (e.g., in a shell).

Dialign: diagonal alignment. A method to construct global alignments by greedily selecting a consistent set of pairwise gapless local alignments (anchors). These anchors appear in a dot plot as diagonals.

DCA: divide-and-conquer alignment. A framework to recursively cut down the sequences to be aligned such that an (near to) optimal alignment path can be found when concatenating the alignments of the substrings. The heuristic is very elaborated such that the algorithm in general is still treated as exact.

MSA: multiple sequence alignment. The efficient implementation of the exact multiple alignment with the Carrillo-Lipman heuristics has the same name.

NJ: Neighbor-Joining. A method to infer phylogenetic trees from distances that come close to a metric. An additional step has to be performed to rate-correct the distance matrix before in each round the pair of neighbors can be identified.

NP-hard: Non-Polynomial. NP-hard is a complexity class for problems for which a solution cannot be found by a polynomial time algorithm. The National Institute of Standards and Technology (NIST) gives the more formal definition: *"The complexity class of decision problems that are intrinsically harder than those that can be solved by a nondeterministic Turing machine in polynomial time."*

OF: objective function. A function that assigns scores which then are optimized by an algorithm. Multiple alignment algorithms use objective functions that capture the similarity of the aligned strings or substrings.

SP: sum-of-pairs method. Sums up the substitution scores of all possible pairwise combinations of sequence characters in one column of a multiple alignment.

T-Coffee: tree-based consistency objective function for alignment evaluation and errr. An alignment strategy that combines global and local alignment in a weighting scheme.

UPGMA: unweighted pair group method using arithmetic averages. A distance-based tree reconstruction method that assumes ultrametric distances. When fusing two nodes to a common ancestor, the newly calculated distances are the arithmetic average which is corrected for size of the subtree each joined node spans.

WPGMA: weighted pair group method using arithmetic averages. A distance-based tree reconstruction method that assumes ultrametric distances. When fusing two nodes to a common ancestor, the newly calculated distances are the arithmetic average.

BIOINFORMATICS APPLICATIONS NOTE Vol. 19 no. 12 2003, pages 1592–1593
DOI: 10.1093/bioinformatics/btg197

QAlign: quality-based multiple alignments with dynamic phylogenetic analysis

M. Sammeth[1, 2, 3,*,†], J. Rothgänger[4], W. Esser[1], J. Albert[1], J. Stoye[3] and D. Harmsen[5]

[1]Department of Computer Science II, University of Würzburg, 97074 Würzburg, Germany, [2]International Graduate School of Bioinformatics, Bielefeld University, 33594 Bielefeld, Germany, [3]Genome Informatics, Department of Technology, Bielefeld University, 33594 Bielefeld, Germany, [4]RIDOM bioinformatics, 97082 Würzburg, Germany and [5]Institute for Hygiene, University of Münster, 48149 Münster, Germany

Received on December 11, 2002; revised on February 12, 2003; accepted on March 4, 2003

ABSTRACT

Summary: Integrating different alignment strategies, a layout editor and tools deriving phylogenetic trees in a 'multiple alignment environment' helps to investigate and enhance results of multiple sequence alignment by hand. QAlign combines algorithms for fast progressive and accurate simultaneous multiple alignment with a versatile editor and a dynamic phylogenetic analysis in a convenient graphical user interface.

Availability: QAlign is freely available over the internet at http://www.ridom.de/qalign/. The platform-independent JAVA technology used provides distributions for various operating systems and hardware architectures.

Contact: qalign@ridom.de

INTRODUCTION

The correct alignment of multiple DNA and protein sequences is a fundamental problem in computational biology. Results produced by the commonly used progressive multiple alignment methods can be obtained rapidly but they are highly dependent on the degree of similarity of the input. Simultaneous alignment algorithms synchronize the information in all sequences to construct the multiple alignment and are therefore more sensitive. However, even these optimal alignment layouts may need some manual editing. Furthermore, downstream analyses (e.g. methods to derive phylogenetic trees) are linked dynamically to the multiple alignment. Thus, a stronger interaction between the creation of the alignment and the phylogenetic analysis enables evolutionary trees of high quality to be found.

*To whom correspondence should be addressed.
†Present address: Genome Informatics, Department of Technology, Bielefeld University, 33594 Bielefeld, Germany.

IMPLEMENTATION

Due to its modular and layered structure, our program QAlign may easily be extended to support additional algorithms for both, multiple alignment and phylogenetic reconstruction. Herein we outline the features included in the current version.

Multiple alignment algorithms

QAlign is a new graphical environment integrating multiple features in the construction of the best multiple alignment for a specific set of sequences (FASTA and MSF sequence format supported). The algorithm monitor controls the construction of the multiple alignment where a fast progressive or a more accurate simultaneous approach may be chosen to align the sequences or parts of them (see Fig. 1 right). The heuristics used in the progressive approach (QAlign uses the variant of the MSA protocol, Gupta *et al.*, 1995) of global multiple sequence alignment allow the alignment of even very large data sets. However, the drawback is that the resulting alignment is a fast approximation of the solution (McClure *et al.*, 1994; Hickson *et al.*, 2000).

In addition, QAlign contains an efficient and stable reimplementation of the NCBI's MSA (multiple sequence alignment) program (Gupta *et al.*, 1995). This is based on the simultaneous alignment strategy, an exact algorithm capable of finding the optimal mathematical solution. In addition to the optimizations used in MSA, the divide-and-conquer algorithm DCA (Tönges *et al.*, 1996; Stoye *et al.*, 1997) was used to achieve the simultaneous alignment of larger data sets. The desired quality-time tradeoff ratio for simultaneous alignment construction can be balanced by a slider. Both the progressive and the simultaneous alignment strategy may be used in a complementary manner on the same alignment layout.

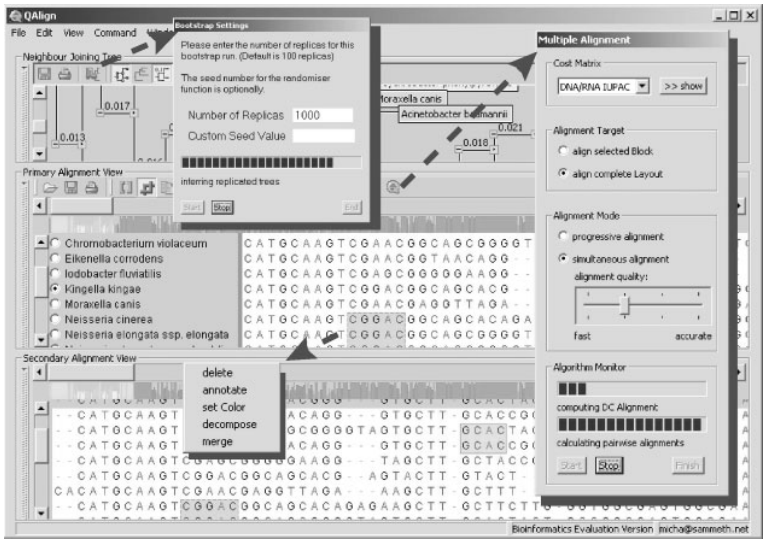


Fig. 1. The graphical user interface of QAlign: the neightbour joining tree is updated dynamically (top) different algorithms and their parameters are accessible by the algorithm monitor (centre-right) and context menus support the editing functions for each block (bottom).

Alignment editor features

After aligning the sequences, the graphical editor of QAlign provides features to analyse the result and modify the multiple alignment layout (as in Fig. 1, bottom). Gaps may be inserted or deleted and marked blocks may be moved within the alignment providing that the aligned sequences have the same length. An immediately updated consensus sequence with coloured bars shows the matching ratio of each column. These bars represent the conservation of different clusters across the alignment layout. They are also displayed as a bird's eye view underneath a scrollbar thus allowing easy navigation to areas of low similarity. A secondary view is provided which may be used to extend the editor capabilities on one alignment or to compare two different multiple alignment layouts.

Dynamic phylogenetic analysis

A dynamic phylogenetic tree view makes visible the consequences of a change in the alignment with regard to the phylogenetic relationship (see Fig. 1, top) where branch lengths may change and nodes may swap according to the neighbour-joining method (Saitou and Nei, 1987). The tree may also be bootstrapped at any time to reveal its current stability. Thus, a phylogenetic reflection of the dynamics of the multiple alignment layout is obtained.

A variety of visual rearrangements is provided for the tree (e.g. subtrees may be collapsed or rearranged). Finally, the phylogenetic tree may be exported, either to a vectorial data format for drawing tools (SVG) or to a common format used by tree plotters (Newick).

CONCLUSION

QAlign provides a practical solution for the creation of refined multiple alignments: layouts produced by various algorithms may be used as a starting-point for changes done by hand, while the phylogenetic consequences are visualised on the fly. Furthermore, the comparison of multiple alignments is made easier because of the two alignment views integrated in the user interface of QAlign.

REFERENCES

Gupta,S.K., Kececioglu,J.D. and Schäffer,A.A. (1995) Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Comp. Biol.*, **2**, 459–472.

Hickson,R.E., Simon,C. and Perrey,S.W. (2000) The performance of several multiple-sequence alignment programs in relation to secondary-structure features for an rRNA sequence. *Mol. Biol. Evol.*, **17**, 530–539.

McClure,M.A., Vasi,T.K. and Fitch,W.M. (1994) Comparative analysis of multiple protein-sequence alignment methods. *Mol. Biol. Evol.*, **11**, 571–592.

Saitou,N. and Nei,M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, **4**, 406–425.

Stoye,J., Moulton,V. and Dress,A.W. (1997) DCA: an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment. *Comput. Appl. Biosci.*, **13**, 625–626.

Tönges,U., Perrey,S.W., Stoye,J. and Dress,A.W.M. (1996) A general method for fast multiple sequence alignment. *Gene*, **172**, GC33–GC41.

BIOINFORMATICS

Vol. 19 Suppl. 2 2003, pages ii189–ii195
DOI: 10.1093/bioinformatics/btg1077

Divide-and-conquer multiple alignment with segment-based constraints

Michael Sammeth [1,*], Burkhard Morgenstern [2] and Jens Stoye [1]

[1]Bielefeld University, Department of Genome Informatics, Technical Faculty, P.O. 10 01 31, 33594 Bielefeld, Germany and [2]University of Göttingen, Department of Bioinformatics, Institute of Microbiology and Genetics, Goldschmidtstr. 1, 37077 Göttingen, Germany

Received on March 17, 2003; accepted on June 9, 2003

ABSTRACT

A large number of methods for multiple sequence alignment are currenty available. Recent benchmarking tests demonstrated that strengths and drawbacks of these methods differ substantially. *Global* strategies can be outperformed by approaches based on *local* similarities and vice versa, depending on the characteristics of the input sequences. In recent years, *mixed* approaches that include both global and local features have shown promising results. Herein, we introduce a new algorithm for multiple sequence alignment that integrates the global *divide-and-conquer* approach with the local *segment-based* approach, thereby combining the strengths of those two strategies.

Contact: micha@sammeth.net

1 INTRODUCTION

Automatic generation of multiple alignments is a central task of computational biology. Although diverse methods are now available, no final solution applicable in all possible alignment situations has been found Notredame (2002). Traditionally, there exist two opposed strategies of alignment construction, one creating *global* alignments and the other one detecting *local* similarities among the input sequences.

For global alignment, *simultaneous* approaches create alignments by synchronising the information of all input sequences in a *k*-dimensional lattice. Although highly elaborated algorithms have been developed to narrow regions of interest within this lattice Gupta (1995); Tönges *et al.* (1996), these approaches are computationally expensive so that their application is limited. For this reason, alternative approaches have been developed where the multiple alignment problem is reduced to a series of pairwise *profile* alignments Feng and Doolittle (1987); Higgins and Sharp (1988); Taylor (1988); the most popular of these *progressive* methods is CLUSTAL W Thompson *et al.* (1994). However, a serious drawback of this technique is that the resulting multiple alignments crucially depend on the *order* in which the profile alignments are carried out.

To cope with more locally related sequence sets, a number of alternative approaches have been proposed that focus on locally related segments of the sequences Depiereux *et al.* (1997); Morgenstern *et al.* (1996); Schuler *et al.* (1991); Vingron and Argos (1991). These approaches are superior to more traditional strategies in situations where large gaps need to be inserted into the alignment and for data sets that are evolutionarily distantly related. However, they may be outperformed by global methods where sequence sets are related over their entire length Lassmann and Sonnhammer (2002); Thompson *et al.* (1999).

Obviously, it is highly desirable to have alignment algorithms performing well on both, globally and locally related sequences. Notredame *et al.* proposed an approach where both, local and global alignment information, is pairwisely preprocessed and extended to the multiple context in a heuristic solution of the maximum weight trace problem Kececioglu (1993). Biasing those preprocessed similarities improved the results of standard progressive alignment, and the resulting procedure has been implemented in the program T-COFFEE Notredame *et al.* (2000). Moreover, Myers *et al.* developed an algorithm for progressive multiple alignment with *constraints* Myers *et al.* (1996). Herein, we introduce an algorithm that performs simultaneous multiple alignment under constraints given by pre-calculated local sequence similarities. In our implementation, we combine the global divide-and-conquer algorithm DCA Stoye (1998) with the local segment-based program DIALIGN Morgenstern (1999). We evaluate this mixed method and compare its results to both of the native protocols and to other successfull alignment methods (i.e. T-COFFEE and CLUSTAL W).

*To whom correspondence should be addressed.

2 TECHNICAL BACKGROUND

A *global alignment* of a family of k sequences $S = (s_1, s_2, \ldots, s_k)$ over a finite alphabet Σ can be defined as a $k \times m$ matrix A with entries in an extended alphabet $\Sigma^* = \Sigma \cup \{-\}$, such that ignoring the blank characters, the pth row reproduces sequence s_p and there is no column consisting exclusively of blanks. A maximal run of adjacent blank characters in a row is called a *gap*. Further, an alignment A can be represented as a source-to-sink path in a k-dimensional *alignment graph*.

In contrast, a *local* alignment aligns only *substrings* of the input sequences. Most frequently, local alignments are used in pairwise sequence comparison ($k = 2$), especially for database searching. A special type of local alignments is the *gap-free* pairwise alignment, by which two substrings of equal length are matched. Note that these substrings are usually expected to share a certain degree of similarity but, in general, they are not required to be identical. Ungapped local alignments between two sequences are sometimes called *fragments* or *diagonals*. Moreover, such gap-free pairwise alignments are also used in the context of multiple sequence comparison. Here, a fragment f is uniquely determined by the involved sequences s_p and s_q, the starting points i and j of the substrings and the fragment length ℓ. Therefore, we will use the shorthand notation $f = ([p, i], [q, j], \ell)$.

Before we will introduce our new fragment-based divide-and-conquer global alignment algorithm, we outline both of the strategies involved, multiple alignment construction based on pairwise fragments and the divide-and-conquer algorithm for simultaneous multiple alignment.

2.1 Segment-based multiple alignment

In the segment-based program DIALIGN, a *weighting* function is defined on the set of all possible fragments, and the program tries to find a *consistent* collection of fragments with maximum total weight Morgenstern (1999). A set F of fragments is called *consistent* if there exists a *global* alignment A_F such that all segment pairs $f \in F$ are aligned by A_F; in this case, we say that A_F *realizes* F. A non-consistent set of fragments is shown in Figure 1.

For multiple alignment, DIALIGN integrates fragments *greedily* into a consistent set F that defines an alignment of the input sequences. To check if a fragment f can be included into F, the algorithm uses so-called *transitivity frontiers*, a data structure first introduced by Abdeddaïm Abdeddaïm (1997). Let us consider the set X of all *positions* x in a sequence family $S = (s_1, s_2, \ldots, s_k)$ where position $x = [p, i]$ corresponds to the ith character in sequence s_p. A collection F of fragments induces a *quasi-partial order relation* $\preceq_F$ on the set X where $x \preceq_F y$ means that x is to the left-hand side or in the same column as y in *every* alignment A_F that realizes F. Formally, the transitivity frontiers are defined as

$$Pred_F(x, p) = \max\{j : [p, j] \preceq_F x\},$$

$$Succ_F(x, p) = \min\{j : x \preceq_F [p, j]\}.$$

In other words, $Pred_F(x, p)$ ($Succ_F(x, p)$) is the index of the right-most (left-most) character in sequence s_p that is to the left (right) of x with respect to the relation $\preceq_F$. These frontiers can be used to decide which residues of the sequences are still alignable without leading to inconsistencies with F Abdeddaïm and Morgenstern (2001), see Figure 2. If the referred set of fragments is obvious, we will drop the index F in $Pred_F(x, p)$ and $Succ_F(x, p)$.

2.2 Divide-and-conquer hyperspace alignment

For a given family of input sequences $S = (s_1, s_2, \ldots, s_k)$, the *divide-and-conquer approach* to simultaneous multiple sequence alignment (DCA) Tönges *et al.* (1996) firstly splits each sequence $s_p \in S$ at a *cut position* c_p. In order to obtain good cut positions, Tönges *et al.* Tönges *et al.* (1996) suggest to use pairwise matrices $C_{p,q}$ for all $s_p, s_q \in S$ that store for each cell (i, j) of the dynamic-programming matrix the *additional alignment costs* that are incurred if the alignment graph is *forced* to leave the optimum and pass the cell (i, j). These matrices are calculated using standard *dynamic programming* procedures. Different heuristics have been described to find families of cut positions $(c_1, c_2, \ldots, c_k)$ that minimise the additional costs in all sequence pairs Stoye (1998). The cutting procedure is recurred until a certain *stop size* of the cut sequences is no longer exceeded. Then, the obtained *atomic* subsequences are aligned optimally and the results are concatenated to form a complete global alignment A.

Within the DCA protocol, the optimal alignment of the atomic sets of subsequences is achieved by applying the *simultaneous alignment* strategy as described for the MSA algorithm Gupta (1995). The latter applies a simultaneous alignment algorithm which basically conquers a k-dimensional (*hyperspace*) lattice using the forward dynamic programming technique concurrently on all sequences in S to find an optimal multiple alignment according to the scoring function. However, heuristics are used to reduce the search effort, e.g. the *Carrillo-Lipman* heuristics Carrillo and Lipman (1988): a progressive alignment is pre-computed for the input set to derive an upper bound on the alignment costs in all projected sequence pairs. Afterwards, pairwise matrices of *total alignment costs* $D_{p,q}$, which are again calculated by dynamic programming procedures, are applied to exclude projections of hyperspace cells from the optimal solution. The final dynamic programming procedure conquering the hyperspace lattice then is sped up by a branch-and-bound procedure skipping all cells which contain an excluded pairwise projection.

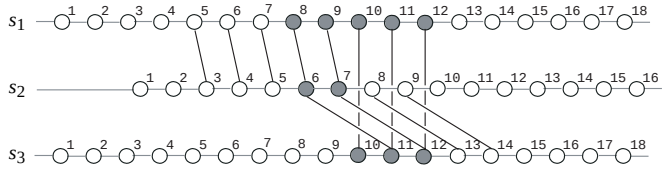


Fig. 1. A set of fragments $f_1, \ldots, f_N$ may be *inconsistent*, i.e., it may not be possible to include all fragments simultaneously in a single multiple alignment. In our example, the three fragments $f_1 = ([1, 5], [2, 3], 5)$, $f_2 = ([2, 6], [3, 11], 4)$ and $f_3 = ([1, 10], [3, 10], 3)$, shown as lines between the aligned indices, would lead to contradicting alignment of the positions marked in grey in a global multiple alignment.

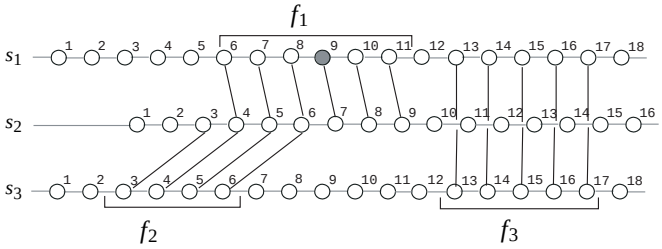


Fig. 2. Example for transitivity frontiers of position [1,9] (marked grey) enforced by the set of fragments $F = \{f_1, f_2, f_3\}$ with $f_1 = ([1, 6], [2, 4], 6)$, $f_2 = ([2, 3], [3, 3], 4)$, $f_3 = ([1, 13], [3, 13], 5)$. For s_3, $Pred([1, 9], 3) = 6$ and $Succ([1, 9], 3) = 13$ are induced. In the special case of aligned positions, transitivity frontiers coincide, for example in s_2, a single position is specified by $Pred([1, 9], 2) = Succ([1, 9], 2) = 7$.

3 THE ALGORITHM

On a high level, our algorithm proceeds as follows: in a first step, we apply the segment-based alignment in order to obtain a consistent set F of fragments representing a framework for further refinement. These fragments—or a suitable subset of them—are used as *constraints* in the second step of the algorithm. Here, we apply the divide-and-conquer method to complete the alignment for those regions that are not aligned by the fragments in F. More precisely, divide-and-conquer computes a (near-)optimal multiple alignment A under the additional constraint that A realises the set F. To control the influence of the fragments in F on the final alignment, we apply a cutoff threshold T and accept only those fragments $f \in F$ that have *overlap weights* exceeding T (see Morgenstern *et al.* (1996) for a definition of these overlap weights). With $T = 0$, the entire set F is used as constraints, and the final alignment is therefore just a minor modification of the DIALIGN alignment. With higher cutoff values, the influence of the segments is reduced, and the resulting output alignments are more similar to what the original divide-and-conquer algorithm would return. If T is large enough to exclude *all* fragments $f \in F$, we obtain exactly the DCA alignment.

While the computation of the fragments can be performed exactly as in the DIALIGN algorithm (or any other procedure that computes consistent local similarities from a family of sequences), three modifications to the divide-and-conquer algorithm are necessary: (1) in the divide step, the cut positions selected to divide the sequences need to be in accordance with our fragment-induced constraints; and in each conquer step, (2a) the heuristic multiple alignment used to calculate the Carrillo-Lipman bounds and (2b) the simultaneous multiple alignment procedure carried out to compute an alignment A_F within these bounds must respect the constraints given by F. These modifications are detailed in the following two subsections.

3.1 The *divide* step with constraints

Our first modification to the original DCA algorithm concerns the way we compute the *additional-cost* matrices that are used to determine cut positions for the sub-alignments. The consistency constraints prevent certain pairs of residues from being aligned to each other. The corresponding graphs in the alignment matrices must therefore be *masked*. To this end, we simply assign infinite additional costs to those edges. Given a sequence pair (s_p, s_q), we need to know those positions j in s_q that are alignable with position $[p, i]$ without leading to inconsistencies; this information must be available for each position $[p, i]$ in sequence s_p and vice versa for each position $[q, j]$ in sequence s_q.

There are two possible scenarios (cf. Figs 2 and 3): the first scenario is that position $[p, i]$ is already (directly or indirectly) aligned with some position $[q, j]$ by the fragments in F. This is the case if and only if we have $Pred([p, i], q) = Succ([p, i], q) = j$. In this case, there is only *one* possible edge in the respective pairwise alignment matrix leading to cell (i, j), namely the diagonal edge coming from position $(i - 1, j - 1)$, see Figure 3, left. The two edges coming from $(i - 1, j)$ and $(i, j - 1)$ have to be excluded—together with all edges to the left and above this edge, as they would correspond to a gap character aligned to position $[p, i]$, in contradiction to the given fragments that impose that $[p, i]$ is aligned with $[q, j]$. The second scenario is that $[p, i]$ and $[q, j]$ are not (yet) aligned, which is the case if and only if $Pred([p, i], q) < Succ([p, i], q)$ holds (Fig. 3, center). In this case, all positions between $Pred([p, i], q) + 1$ and $Succ([p, i], q) - 1$ can be aligned with $[p, i]$ without leading to inconsistencies. For $Pred([p, i], q) + 1 \leq j \leq Succ([p, i], q) - 1$ (Fig. 3, right), the cell (i, j) in the dynamic programming matrix can be reached from all three possible positions. In addition, cell $(i, Pred([p, i], q))$ can be reached from above (but not from the left or from the top-left) and cell $(i, Succ([p, i], q))$ can be reached from the left (but not from above or from the top-left), see Figure 3, center and right. After this masking procedure, the original procedure used in DCA can be employed to identify suitable cut positions for the *divide* step.

3.2 The *conquer* step with constraints

In the Carrillo-Lipman approach, all optimal pairwise alignments as well as a heuristic multiple alignment of the input sequences are computed in order to determine *boundaries* for the pairwise *projections* of an optimal multiple alignment to the respective pairwise comparison matrices Carrillo and Lipman (1988). In our constrained scenario, we need to impose the consistency constraints both to the optimal pairwise alignments and to the heuristic multiple alignment. The pairwise alignments are calculated as explained in Section 3.1. For the heuristic multiple alignment, where we use a progressive approach with *profile alignments* described in Gupta (1995), in addition to alignments of individual sequences, a very similar consistency-constrained approach can be applied, too. To determine which positions are alignable between two profiles, the transitivity frontiers of all involved sequence pairs are considered. The alignable regions for the profiles are then given as the *intersections* of the alignable regions for the individual sequence pairs.

Once the pairwise alignments and the heuristic multiple alignment are computed, restrictions for the *projections* of an optimal multiple alignment to all pairwise comparison matrices can be calculated. In our approach, we have additional constraints derived from our transitivity frontiers. The regions allowed for the projections to the pairwise matrices are simply given as the intersections between the regions calculated using the Carrillo-Lipman boundaries and the regions defined by our transitivity frontiers. An optimal multiple alignment realising F is finally computed in the space that is defined by these combined pairwise restrictions, similar to the original Carrillo-Lipman approach.

4 RESULTS AND DISCUSSION

To evaluate the performance of our new algorithm, we benchmarked it against DIALIGN, DCA, CLUSTAL W and T-COFFEE. For DCA, we used a re-implementation that employs a more stable version of MSA. For all programs, we used their default parameter settings together with the BLOSUM-62 substitution matrix. Our algorithm was implemented in JAVA and the program was tested with a hotspot capable runtime environment (JDK1.4.1 with a maximum of 3 GB heap memory). The main benchmarks were performed on a SUN workstation (SUN Fire 880, 750 MHz, 32 GB main memory). Data for benchmarking was derived from the reference alignments in BAliBASE, version 1 Thompson *et al.* (1999), where reference alignments are sorted into five groups according to the characteristics of the input, respectively. Within the first three groups, the sequences of the input share a variant degree of global affinity, but similarity drops from group 1 to group 3. Group 4 contains alignments with N/C-terminal extensions, while in group 5 sequences with internal insertions have been collected. To compare the computed alignments to the BAliBASE benchmark alignments, we used the program `aln_compare` written by C. Notredame (personal communication).

Table 1 summarizes the main results. For each group of alignments from the BAliBASE data set, the average percentage identity with the reference alignments is given, taking into account structural columns as annotated in BAliBASE. A basic version of our program denoted by *Mixed* $T = 0$ in Table 1 uses the entire set of fragments returned by DIALIGN as constraints for the divide-and-conquer alignment. In addition, we performed runs where the cutoff value T was set to higher values, such that only fragments with overlap weights above T were used. For comparison, we also ran two popular multiple alignment programs from the literature, CLUSTAL W Thompson *et al.* (1994) and T-COFFEE Notredame *et al.* (2000).

As can be seen from Table 1, the strengths of DCA and DIALIGN differ substantially. The mixed approach shows characteristics of both strategies, refining the segment-based local alignment computed by DIALIGN with the divide-and-conquer hyperspace alignment stragegy (DCA). Problems arise in group 2 and group 3 where neither DIALIGN can provide good local similarities,

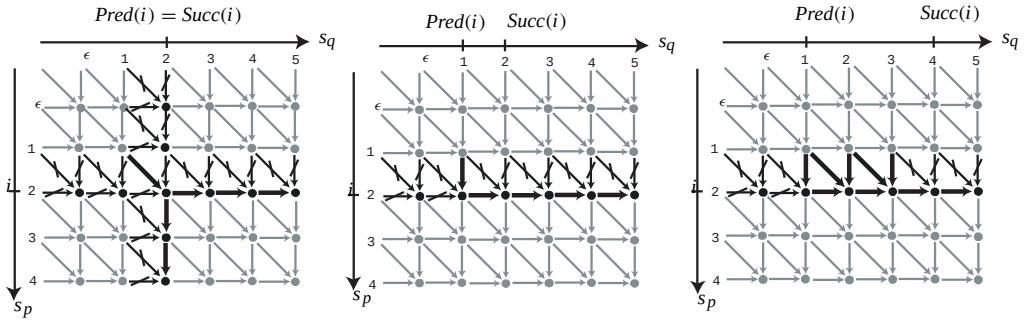


Fig. 3. Three examples of transitivity frontiers for $i = 2$ in a two dimensional dynamic programming matrix of s_p and s_q. Black dots represent cells within the matrix of which incoming edges are affected by the transitivity frontiers considered here.

Table 1. Percent identity of the results produced by the various algorithms with the reference alignments given by BAliBASE (according to the column score of trusted regions). The alignment count is given in parentheses after each group in the top row

Algorithm	Group 1 (83)	Group 2 (23)	Group 3 (12)	Group 4 (13)	Group 5 (11)
DCA	79.27	29.47	37.47	58.81	78.93
Mixed $T = 10$	76.03	26.06	35.71	71.52	72.45
Mixed $T = 7$	76.11	24.66	33.84	73.74	80.54
Mixed $T = 5$	76.26	25.49	34.04	74.82	85.08
Mixed $T = 3$	74.39	24.86	35.73	78.93	83.74
Mixed $T = 0$	73.07	25.39	35.49	75.29	80.38
DIALIGN	71.95	25.13	35.10	74.66	80.38
CLUSTAL W	79.53	32.91	48.72	74.02	67.84
T-COFFEE	76.86	36.89	49.94	81.28	86.25

nor does the scoring function maximized by DCA yield results close to the reference alignments. However, in group 1 and in the last two groups, the best scoring result of the mixed strategy is very close to the T-COFFEE results. Note that the values given are arithmetic averages over all alignments in a group such that single alignments within a group can score even better than with T-COFFEE. Because of the global affinity within group 1, the global strategies perform very good there (see CLUSTAL W, DCA). In contrast, exclusively global methods perform rather poorly in group 4 and group 5, whereas the mixed strategy can successfully integrate the local fragments into the global alignment (i.e. it scores better than any of the underlying strategies alone).

Concerning the influence of the cutoff parameter on the quality of the resulting alignment, it can be seen from Table 1 that the best value for T depends on the characteristics of the input data. In the first three groups of BAliBASE where DCA is superior to DIALIGN, a high cutoff prevents spurious fragments from deteriorating the final alignment and increases the quality of the program output. For groups 4 and 5, however, DIALIGN fragments tend to improve the alignment produced by DCA. Here, a lower cutoff leads to better results.

Table 2 reports on the running time for the programs we analyzed. Alignments that did not terminate within 24 hours were considered as *failures*. The reason for this phenomenon is that, even with efficient heuristics, the search for high-quality cut positions and the subsequent hyperspace alignment may be time-consuming for large or badly matching sequence families. Our results show that if input sequences are easy to align, the mixed protocol takes slightly more time than the two underlying algorithms together. This is because additional CPU time is necessary to mask all matrices used in DCA according to the set of fragments found by DIALIGN. However, the time spent for this purpose quickly pays off for input sequences that are difficult to align such as in groups 2–5. Here all mixed results show less failures and are aligned faster than with the original DCA algorithm. Fragments can, of course, slow down the search for optimal cut points if they are not in agreement with the global optimal alignment (see failure counts in Table 2, Mixed $T = 10$). In general, however, the fragments from the local alignment reduce the part of the hyperspace that needs to be considered by DCA, thereby speeding up the divide-and-conquer

Table 2. Time consumption in seconds and number of failed runs (in parentheses) for the various programs summed up for each BAliBASE group. The total number of alignments in each group is given in parentheses in the top row

Algorithm	**Group 1 (83)**		**Group 2 (23)**		**Group 3 (12)**		**Group 4 (13)**		**Group 5 (11)**	
DCA	136.33	(0)	119430.51	(1)	307753.98	(5)	9918.90	(2)	42252.48	(2)
Mixed $T = 10$	766.02	(0)	8942.64	(1)	1909.89	(1)	4427.80	(1)	42411.52	(1)
Mixed $T = 7$	680.62	(0)	8634.20	(0)	1475.39	(2)	1969.15	(1)	42175.50	(1)
Mixed $T = 5$	445.24	(0)	3435.06	(0)	1360.25	(2)	459.20	(1)	42051.03	(0)
Mixed $T = 3$	487.88	(0)	3014.28	(0)	1360.90	(2)	324.16	(1)	17202.66	(0)
Mixed $T = 0$	435.00	(0)	1652.18	(0)	1279.20	(0)	289.98	(0)	276.57	(0)
DIALIGN	136.67	(0)	525.97	(0)	386.45	(0)	141.51	(0)	94.49	(0)
CLUSTAL W	140.19	(0)	445.47	(0)	136.28	(0)	71.25	(0)	50.67	(0)
T-COFFEE	447.68	(0)	4614.28	(0)	2485.24	(0)	913.01	(0)	550.33	(0)

Table 3. Column scores and running time (in parentheses) for different algorithm runs in group 5 of the BAliBASE

data set	**Mixed $T = 5$**		**Mixed $T = 3$**		**Mixed $T = 0$**		**T-COFFEE**	
1eft	71.4	(5.03)	71.4	(5.30)	71.4	(5.66)	71.4	(15.94)
1ivy	94.7	(21.19)	92.8	(21.39)	77.8	(20.85)	75.6	(20.40)
1qpg	100.0	(73.35)	100.0	(74.60)	100.0	(73.82)	100.0	(131.08)
1thm1	66.7	(19.83)	66.7	(18.43)	55.6	(10.83)	83.3	(37.49)
1thm2	76.7	(4.01)	76.7	(3.71)	88.3	(4.03)	88.3	(12.85)
2cba	100.0	(11.23)	100.0	(9.35)	100.0	(9.22)	96.7	(17.52)
kinase1	93.5	(4.87)	80.6	(4.52)	80.6	(5.83)	91.0	(78.98)
kinase2	66.7	(76.55)	66.7	(79.93)	66.7	(31.64)	100.0	(5.37)
kinase3	81.2	(83.93)	81.2	(94.08)	81.2	(77.56)	80.6	(6.23)
S51	85.0	(41744.78)	85.0	(16884.04)	62.6	(32.90)	88.9	(62.90)
S52	100.0	(6.28)	100.0	(7.32)	100.0	(4.24)	72.9	(161.57)

approach and causing fewer fails. Obviously, this time gain is reduced if more of the fragments are filtered out. Nevertheless, in most instances the mixed strategy is still faster than T-COFFEE which needs additional CPU time to preprocess and bias local information of the input and therefore takes about ten times the running time of CLUSTAL W. To our knowledge this is the first time that a simultaneous alignment strategy is about as time efficient as progressive heuristics. Moreover, note that the mixed strategy is implemented in JAVA as opposed to CLUSTAL W and T-COFFEE that are written in C, such that there is probably an even bigger algorithmic time gain than reflected by the time measurements shown in Table 2.

Finally, Table 3 gives an overview of the quality and running time for all 11 individual alignments of BAliBASE group 5. In the specific alignment runs, a general trend of quality and time gain for different cutoff values is hard to observe. However, in a few cases the time effort is sensitively triggered by T (e.g. the running time for S51 runs can be reduced from 11.6 to 4.7 hours by increasing the number of fragments from $T = 5$ to $T = 3$ while the score remains the same). Although both protocols inherit from global and local alignment methods, results show that the strengths of mixed DIALIGN–DCA strategy differ from the ones of T-COFFEE (e.g. the kinase2 set is aligned correctly by T-COFFEE while the mixed protocol just reaches 2/3 identity with the reference, whereas for the S52 set the situation is the reverse). Thus, carefully chosen fragments are essential in combining local and global alignment strategies.

As could be demonstrated by the BAliBASE benchmark results, the mixed alignment strategy successively combines the strengths of DIALIGN (local) and DCA (global) multiple alignment. On the one side, the mixed method takes full advantage of the hyperspace alignment, while it reduces the computational time (and space) requirements necessary for this method (see Table 2). In most reference groups, the average results can reach a value close to the T-COFFEE algorithm (see Table 1), and they are generally faster calculated. Single alignments within a group of the BAliBASE reference set are even outperforming results of the T-COFFEE algorithm (see Table 3).

Furthermore, different values empirically tested for the cutoff T of fragment scores were found to show peaks in alignment quality for each group in the BAliBASE

reference. Hence, in further work we will concentrate on replacing the naive threshold-based fragment filter by a weighting scheme that biases the score of each fragment during the global alignment construction. Our hope is that we can achieve a simultaneous dynamic programming algorithm that considers the extended information of local fragments but is not misled if the similarities found turn out to be incorrect for a global solution.

ACKNOWLEDGEMENTS

We would like to thank Cedric Notredame for providing a reliable program for the comparison of test alignments to the BAliBASE reference alignments.

REFERENCES

Abdeddaïm,S. (1997) Incremental computation of transitive closure and greedy alignment. *Proceedings of 8th Annual Symposium on Combinatorial Pattern Matching (CPM 1997)*, volume 1264 of *LNCS*, pp. 167–179.

Abdeddaïm,S. and Morgenstern,B. (2001) Speeding up the DIALIGN multiple alignment program by using the 'greedy alignment of biological sequences library' (GABIOS-LIB). *Proceedings Journées Ouvertes: Biologie, Informatique, Mathématiques (JOBIM 2000)*, volume 2066 of *LNCS*, pp. 1–11.

Carrillo,H. and Lipman,D. (1988) The multiple sequence alignment problem in biology. *SIAM J. Applied Math.*, **48**, 1073–1082.

Depiereux,E., Baudoux,G., Briffeuil,P., Reginster,I., Boll,X.D., Vinals,C. and Feytmans,E. (1997) Match-Box server: a multiple sequence alignment tool placing emphasis on reliability. *CABIOS*, **13**, 249–256.

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

Gupta,S., Kececioglu,J. and Schäffer,A. (1995) Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Comp. Biol.*, **2**, 459–472.

Higgins,D. and Sharp,P. (1988) CLUSTAL—a package for performing multiple sequence alignment on a microcomputer. *Gene*, **73**, 237–244.

Kececioglu,J. (1993) The maximum weight trace problem in multiple sequence alignment. *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching (CPM 1993)*, volume 684 of *LNCS*, pp. 106–119.

Lassmann,T. and Sonnhammer,E. (2002) Quality assessment of multiple alignment programs. *FEBS Lett.*, **529**, 126–130.

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

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

Myers,G., Selznick,S., Zhang,Z. and Miller,W. (1996) Progressive multiple alignment with constraints. *J. Comp. Biol.*, **3**, 563–572.

Notredame,C. (2002) Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, **3**, 131–144.

Notredame,C., Higgins,D. and Heringa,J. (2000) T-COFFEE: a novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, **302**, 205–217.

Schuler,G.D., Altschul,S.F. and Lipman,D.J. (1991) A workbench for multiple alignment construction and analysis. *Proteins: Struct. Funct. Genet.*, **9**, 180–190.

Stoye,J. (1998) Multiple sequence alignment with the divide-and-conquer method. *Gene*, **211**, GC45–GC56.

Taylor,W.R. (1988) A flexible method to align large numbers of biological sequences. *J. Mol. Evol.*, **28**, 161–169.

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

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

Thompson,J., Plewniak,F. and Poch,O. (1999) A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Res.*, **27**, 2682–2690.

Tönges,U., Perrey,S., Stoye,J. and Dress,A. (1996) A general method for fast multiple sequence alignment. *Gene*, **172**, GC33–GC41.

Vingron,M. and Argos,P. (1991) Motif recognition and alignment for many sequences by comparison of dot-matrices. *J. Mol. Biol.*, **218**, 33–43.

PROTEINS: Structure, Function, and Bioinformatics 64:263–274 (2006)

Global Multiple-Sequence Alignment With Repeats

Michael Sammeth* **and Jaap Heringa**
Centre for Integrative Bioinformatics (IBIVU), Vrije Universiteit, Amsterdam, The Netherlands

ABSTRACT **Repeating fragments in biological sequences are often essential for structure and function. Over the years, many methods have been developed to recognize repeats or to multiply align protein sequences. However, the integration of these two methodologies has been largely unexplored to date. Here, we present a new method capable of globally aligning multiple input sequences under the constraints of a given repeat analysis. The method supports different stringency modes to adapt to various levels of detail and reliability of the repeat information available. Proteins 2006;64:263–274.**

INTRODUCTION

One of nature's most striking inventions is repeated use of sequence motifs throughout information-carrying biopolymers such as DNA, RNA, and protein molecules. In sequence analysis, we call such repeated substrings simply *repeats*, which sometimes constitute most of the respective biomolecule in which they are contained. Indeed, half the human genome is known to consist of repeating fragments. The spectrum of repeat functions in DNA includes structural, replicational, and viral infection mechanisms.

In proteins, multiple copies of the same basic *motif* can, for example, determine the three-dimensional (3D) structure or bear essential functionality for protein–protein or protein-DNA interactions. In both, DNA or protein sequences, the repeat strings may either be exact or degenerated (i.e., "approximate repeats" in literature), and divergence of repeats can be particularly significant in proteins. Furthermore, the repeat copies may be in tandem positions (i.e., two repeats immediately following each other in the string) or more arbitrarily dispersed over the sequence. These properties make repeats a highly interesting research topic in sequence analysis, and a vast number of problems on repetitive structures have been formulated in the literature, such as the problem of repeat detection (see below), distance-based comparison[1] and evolutionary history.[2]

Given the widespread occurence of repeats throughout the genomes of the major kingdoms, it is surprising to see that relatively few efforts have been made to incorporate information about sequence repeats into sequence alignment methods. On the other hand, repeats are known to be a major obstacle in the generation of informative alignments. Recently, a survey about existing alignment methods concluded that "repeated sequences (in tandem or not) are renowned for confusing all existing MSA methods."[3] What is it about the repeats that cannot be handled by the alignment strategies we have? One problem with repeats is that because of their evolutionary mechanisms, repeats often interrupt informative subsequences, preventing those effectively from being aligned correctly. Generally, there are two ways to align sequences: while a *global* alignment covers all sequence positions, a *local* alignment aligns similar substrings of the input sequences and disregards the remaining sequence regions. When dealing with a sufficiently high number of adjacent copies, global alignment methods are usually forced by their scoring schemes to align positions of repeat clusters with positions of the intervening sequence. Even modern global alignment strategies that are successfully incorporating local similarities, such as T-Coffee[4] and Dialign,[5] can only tolerate accumulations of repeats up to a certain level.

Over the last decade, much attention has been devoted to both automated and manual extraction of repeats from biological sequences. Some databases have been set up to store repetitive sequences. Also, BAliBase,[6] one of the standard alignment benchmark databases, contains reference sequences with repeats. Furthermore, a number of automated methods for repeat detection have been developed. Designed for a specific problem (e.g., exact or degenerated repeats, in tandem position or not, in a nucleic acid or protein sequence, …), these repeat extraction methods can be based on suffix trees such as Reputer[7] or on local alignment, e.g., Repro,[8] Radar,[9] and Trust.[10] Although some of these methods have been in existence for >10 years, no databases or tools developed for repeats have been integrated within any global sequence alignment method. In this work, we address this problem and describe a novel procedure to align repeated sequences, in which information about repeats is merged with similarities deduced from comparing nonrepetitious areas of the sequences. The evolutionary model used can cope with translocations that interrupt existing sequence motifs. The method finally aligns the sequences in a standard row-column alignment layout, where shuffled elements remain unaligned in keeping with the global alignment requirement. We assess alignments of repeat sequences produced by our new method with those made by two alternative state-of-the-art global alignment methods, Di-

Grant sponsor: Ernst Schering Research Foundation.

*Correspondence to: Michael Sammeth, Genome Informatics, Bielefeld University, PO 10 01 31, 33501 Bielefeld, Germany. E-mail: micha@sammeth.net

Received 25 August 2005; Revised 12 November 2005; Accepted 12 November 2005

Published online 11 April 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/prot.20957

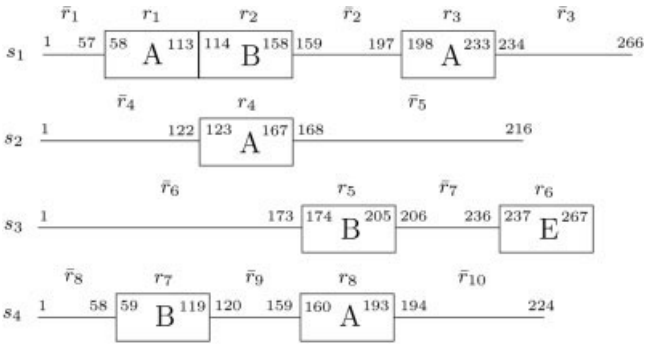


Figure 1. An unaligned sequence set containing some partly repeated motifs (boxes) of different types τ (labeled with capital letters A, B, and E). Note that the repeats R and the intervening areas $\bar{R}$ are numbered consecutively throughout the sequence set. In addition, the start and the end position of each substring are given. Therefore, each repeat can be specified by its sequence number p, the start position i, its length ℓ and its type τ, e.g., $r_2 = (p = 2, i = 114, \ell = 45, \tau = B)$.

align and T-Coffee, which are unaware of the repeats in the sequences.

BACKGROUND

Our method aims to establish a *global alignment* A for a given a family of k sequences $S = \{s_1, s_2, \ldots, s_k\}$ over a finite alphabet Σ and with a set of putative repeats $R = \{r_1, r_2, \ldots, r_l\}$. Formally, A can also be defined as a $k \times m$ matrix with entries in an extended alphabet $\Sigma^* = \Sigma \cup \{-\}$, such that when additional gap characters "$-$" are ignored, row p in the matrix reproduces sequence s_p, and no column in the alignment consists exclusively of gap characters.

However, in an intermediate step, we use the technique of *local alignment*, which aligns only substrings of the input sequences, to find *anchors* for the global alignment. Anchors are strongly conserved subareas of different sequences that later on aid to construct a correct global alignment. They are found by gap-free pairwise local alignment by which two substrings of equal length are aligned. Such anchors $f = ([p,i],[q,j],\ell)$ are uniquely defined by the involved sequences s_p and s_q, the starting points i and j of the aligned substrings and the anchor length ℓ. One of the standard methods to construct a global alignment from such anchors is Dialign.[5] Some aspects of the Dialign algorithm that are relevant for this work are outlined in Appendix A.

Sequences Containing Repeats

In addition to the input sequences $S = \{s_1, s_2, \ldots, s_k\}$, our algorithm allows the specification of a set of constituent repeated or shuffled motifs $R = \{r_1, r_2, \ldots, r_l\}$ contained in these. Throughout this work, repeats are regarded as similar but not necessarily identical substrings that may appear more than once freely dispersed over the sequences and positions (i.e., they do not need to be tandem-repeated, exactly adjacent to each other, etc.) as depicted in Figure 1.

Repeated motifs are characterized by a start position i within a specific sequence $s_p \in S$ and a certain length ℓ. Optionally, we allow the user to divide the repeats into groups of different types τ according to a certain criterion (motif variations, different domains in multidomain proteins, etc.). Therefore, we denote repeats by $r = (p,i,\ell,\tau)$ or in case of only one repeat type by $r = (p,i,\ell)$. The set of all repeats R tokenizes the sequences $s \in S$ into repeat substrings $r \in R$ and *non-repetitious* or *intervening* subsequences $\bar{r} = (p,i,\ell) \in \bar{R}$ in between the repeats.

THE ALGORITHM

Our repeat sensitive alignment takes in addition to the sequences $S = \{s_1, s_2, \ldots, s_k\}$ the repeats information $R = \{r_1, r_2, \ldots, r_l\}$ as input. For each subset of repeats of same type τ, a global alignment A_{R_τ} can be provided as user input or it is computed on-the-fly by divide-and-conquer alignment DCA.[11,12]

The algorithm treats substrings of the input sequences S differentially depending on whether they are contained in R or if they are nonrepetitious (i.e., the substring is included in $\bar{R}$). Basically, for both subsets pairwise anchors f are found, which are then assembled to form a consistent set of high-scoring anchors F with respect to the overlapping weight $olw(f)$ (see Appendix A for definition). In the end, a global alignment layout is produced by submitting all anchors of F as constraints to the constrained divide-and-conquer alignment.[13]

Finding Anchors

To apply the Dialign strategy[5] (Appendix A) of anchor assembly, information of the input sequences has to be tokenized into pairwise anchors $f = ([p,i],[q,j],\ell)$. According to their different nature, this step is performed in another way for the repeated motifs than for intervening sequences (non-motifs).

Anchors in the intervening sequences

True pairwise relations can effectively be overshadowed by short dominating similarities in the repeats. Moreover, an alignment of a subarea of a known repeat to any part of a nonrepeat subsequence has no biological significance, regardless of the score yielded (although we allow a certain unreliability of the information in R, see the relaxed stringency mode described below). To address these problems, we filter out the motifs before we determine the anchors of the intervening sequence areas (Fig. 2).

Thus, we generate for each sequence $s_p \in S$ [Fig. 2(a)] a concatenation of the respective intervening subsequences $\bar{r} = (p,i,\ell)$ by taking out the motifs [Fig. 2(b)]. Then we locally align these sequences by applying the standard Dialign protocol [Fig. 2(c), Appendix A]. The result is a set of pairwise anchors for the intervening sequences. These anchors are then rescored and remapped by reinserting the motifs [Fig. 2(d)].

Anchors derived from the repeats

Figure 3 outlines how information of the global alignment(s) A_{R_τ} are used to obtain anchor-based similarity information of the motifs. All projection pairs (r_v, r_w) from every A_{R_τ} are grouped into anchors by columns containing gap characters. All anchors f_x found for a certain repeat pair (r_v, r_w) then are assigned a weight $w(f_x)$ in the context

Figure 2. Basic outline of finding local similarities in the intervening sequences. While r-areas correspond to motifs, substrings denoted by $\bar{r}$ designate non-motifs. From the initial data set (**a**), all repeat areas $r \in R$ are filtered out (**b**), concatenating the intervening sequences at the || marked positions. **c:** Next, anchors (gray shadowed links between substrings of sequences) are found by applying the standard Dialign protocol. **d:** Finally, the repeat subsequences are reinserted and the weight of all anchors found is recalculated. Note that some of the alignments (gray links) are broken upon reinsertion of the motifs. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

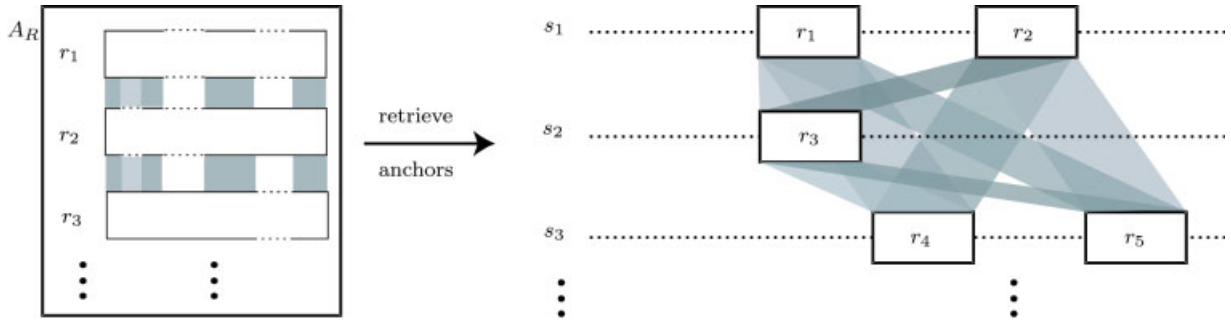


Figure 3. Constructing anchors for the motifs. Gapless anchors f are assigned to each repeat pair as aligned in the global alignment(s) $A_{R\bar{r}}$. The combined weight (r_v, r_w) for the anchor f_{r_v,r_w} is derived by adding up the weights for all gapless local alignments. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

of the original sequences s_p and s_q. Motivated by multiplicating the underlying error probabilities, the logarithmic weights of the anchors are added up to yield the weight of the composed anchor $w(f) = \Sigma_x w(f_x)$.

Mixing the anchors

After the preceding two steps, both sorts of anchors are merged, the ones derived from intervening sequences $\bar{R}$ as well as the ones deduced from the repeat alignment(s) $A_{R\bar{r}}$. To add a certain flexibility for biological facts, our algorithm can operate in two different modes (Fig. 4). First, the so-called *strict mode* [Fig. 4(a)] just merges the repeat anchors (see above section) with nonrepeat anchors. As a result, no part of a designated repeat area may be aligned with an intervening sequence part (strict constraint).

However, to allow the input of repeat information of lower reliability, for instance produced by fully automated repeat prediction, we added a second possibility of aligning

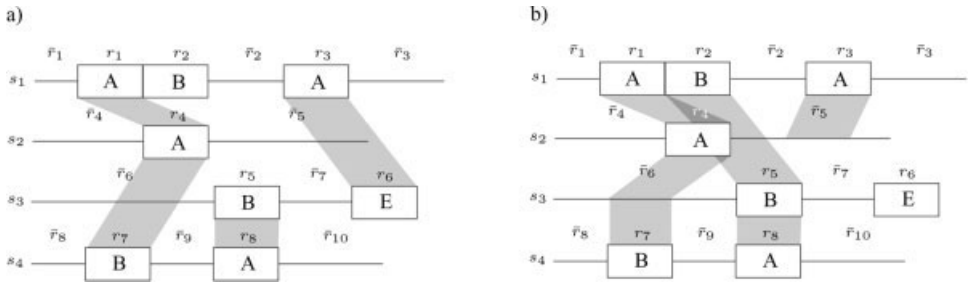


Figure 4. Two stringency modes for the repeat-sensitive alignment. **a:** In the *strict* mode, repeats may be aligned with each other (gray links) and also intervening sequences may only be aligned with other parts of intervening sequences (not depicted). No cross-links between any $r \in R$ and $\bar{r} \in \bar{R}$ are possible. **b:** The relaxed mode allows also areas of the motifs to be aligned with intervening areas.

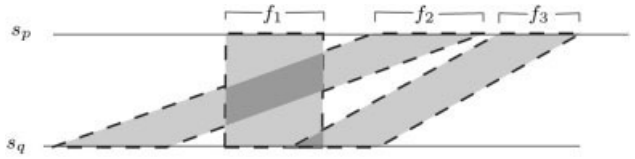


Figure 5. Inconsistent anchors f_1, f_2, and f_3 of a sequence pair (s_p,s_q). Anchor f_1 is contradicting the anchors f_2 (crossing) and f_3 (overlap). Therefore $\{f_1, f_2, f_3\}$, $\{f_1, f_2\}$, $\{f_1, f_3\}$ are inconsistent sets, but $\{f_2, f_3\}$ is consistent.

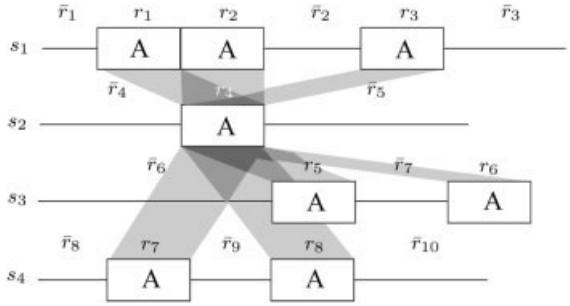


Figure 6. Example of stacked anchors retrieved from the repeat alignment. The number of possible matches grows with the number of repeat copies in the other sequences. According to the definition of $olw(f)$ (Appendix A), repeat r_4 (in the center of the starlike link structure) would receive additional positive weights from all other anchors (the gray links connecting it with the other repeats $\{r_1,\ldots,r_8\}\backslash r_4$). The resulting overlap weights would be falsified (i.e., overestimated according to the number of repeats in the sequence set).

the motifs with *relaxed constraints*. Here, additionally cross-linked anchors between motifs and intervening sequences are searched by applying the standard Dialign protocol on the sequences. The anchors retrieved then are pooled with the local similarities found earlier [Fig. 4(b)].

Resolving Inconsistencies

Within the anchor detection process, inconsistencies are tolerated at two stages. First and foremost, the anchors imposed on the motifs given are not checked for pairwise feasibility: as implied by the definition of a repeat, there may be multiple mutually exclusive possibilities to align the repeated motifs of a sequence pair. When talking about mutually exclusive anchors, we also say the anchors are contradicting each other. However, even with such pairwise contradictions resolved, gaining overall consistency of segments in the context of all input sequences requires further filtering.

Contradicting pairwise matches

When regarding the anchors of a sequence pair $(s_p,s_q) \in S$, some positions may participate in more than one anchor with another sequence (in Fig. 5, the intersection of f_1 and f_3) or contradict another anchor proposed for the same sequence pair (Fig. 5, f_1 and f_2). Such inconsistent sets of anchors are a problem when calculating the overlap weights $olw(f)$, because the same overlapping position if included in multiple anchors biases corresponding anchors it overlaps (see Appendix A for the definition of olw-weights). Especially with repeats, which by definition support each other well, the problem of overrating the overlap weights is crucial (Fig. 6).

To address this problem, we only consider anchors of an optimal pairwise consistent set $\tilde{F}$ when giving additional weights during the calculation of the $olw(f)$. The modified weighting function, denoted by $olw_{\tilde{F}}(f)$, keeps from overestimating the overlap weights of the repeats since the weight of each anchor is increased by at most one other anchor shared with every other sequence. Note that anchors not participating in $\tilde{F}$ are neither excluded from receiving additional weights by overlaps, nor are they discarded. They are only prevented from giving weights to overlapping anchors and so they cannot bias the repeat signal.

Retrieving a global alignment layout

Once the overlapping scores $olw_{\tilde{F}}(f)$ for all anchors f have been computed, inconsistencies caused by transitive links of the anchors between three or more sequences are to be eliminated. To do so, we proceed following the Dialign protocol and greedily integrate the anchors in a set of consistent anchors F. All anchors are iterated according to the $olw_{\tilde{F}}(f)$ assigned to them, starting at the one with the highest overlap weight. Whenever an anchor f is inconsistent with the anchors already contained in F, it is discarded, otherwise it is added to F.

As a result of the protocol, all anchors included in F are consistent (i.e., they can all be incorporated in a single global multiple alignment). In the final step, the *divide-and-conquer multiple alignment with segment-based con-*

straints[13] is applied to positions not yet aligned by the anchors $f \in F$. To be specific, the set F found in the preceding step is submitted as a constraint framework for divide-and-conquer alignment. All hitherto unaligned positions are aligned optimally, and the algorithm terminates by providing a global alignment layout.

RESULTS

In the first part of this section, we compare our novel method with other alignment methods. As reference alignments we use those given by BAliBase (version 2.0, reference group 6),[6] a database for alignment benchmarking. Furthermore, we evaluate different parameters of our method using these alignments. We also provide some detailed comparisons of single alignment case studies to show the advantages of our repeat-sensitive alignment method.

BAliBase

In the second version of BAliBase,[6] the authors included sequence sets containing repeats (reference group 6). As in other groups, the repeat group is subdivided in different subsets according to common features of the data. All these test categories were used in an assessment of our method:

test 1a unique repeat type, same number of repeats

test 1b unique repeat type, variable number of repeats

test 2a variable types, same number and order of repeats

test 2b variable types, same number but varying order of repeats

test 2c variable types, varying number and order of repeats

test 3 variable types of repeats with an additional domain of different type

test 4 motifs and repeats of variable types

test families of proteins including attributes from all of the data sets before

BAliBase provides alignments, where structural information has been superpositioned on the sequences to identify so-called core regions, which are aligned without uncertainty. However, in the BAliBase repeat group, only the repeated substrings are aligned and not the reference data sets over their complete sequence lengths.

Comparison with other alignment methods

Because the reference is only available for the repeat areas, it cannot be evaluated whether intervening regions are aligned correctly. We use the traditional sum-of-pair (SP) score to count correctly aligned positions within the repeats. According to the SP scheme, each character tuple of the test alignment is assigned a score of 1 if it is also aligned in the reference alignment; otherwise its score is 0. The scores of all these character pairs are summed up to the total SP score of the alignment. Applying this, we determine the SP score of aligned repeat positions ($SP_{correct}$) and in the same manner the sum of misaligned repeat positions (SP_{wrong}). Here, a repeat position is considered misaligned, whenever it is aligned with a position within the intervening sequences.

Motivated by traditions in gene finding,[14] we define a measure upon the SP scores, the repeat separation index (SI): $SI = \frac{SP_{correct}}{(SP_{correct}+SP_{wrong})}$. The SI measure gives a basic idea of how well the positions within the repeat areas are aligned: values range from 0.0 (no repeat position aligned correctly, all repeat positions have been aligned with characters outside of repeat areas) to 1.0 (all positions of the repeat areas have been aligned with each other as given by the BAliBase reference). Note that the SI does not represent the correct fraction of the whole alignment, nor does it give any information about the quality of the alignment in the nonrepetitious sequence areas.

Table I compares the new method (RAlign) with other techniques for global alignment (T-Coffee and Dialign). While Dialign[5] uses exclusively local alignments to construct the global alignment, T-Coffee[4] integrates local and global pairwise comparisons before performing the final global alignment. For the comparison, Dialign version 2.2 and T-Coffee version 2.15 were used, both with the default parameters, and the time benchmarks were performed on a Pentium 4 (3.4GHz, 2GB RAM) except for the largest time- or memory-intensive problems, which had to be submitted to a SunFire 880 with 32GB RAM to complete (failures in Table I).

Table I summarizes the BAliBase benchmark tests for the repeat-sensitive alignment (RAlign). The results show the capabilities of the different methods to deal with the difficulties imposed by the alignment characteristics of the corresponding subgroup. Although in general the local aligner Dialign (Appendix A) is the better performer of the two reference methods, in test sets with a more global similarity T-Coffee comes close ("test 2a") or even outperforms ("test 1a") Dialign (i.e., sets with equal number and order of repeats). Within group 1 (repeats of equal subtypes) and group 2 (repeats of variable subtypes), the ratio of correctly aligned repeat tuples $\left(\frac{SP_{correct}}{SP_{wrong}}\right)$ decreases in both reference methods: sequences from subgroup 2a (with equal number of repeats) for instance are aligned better than sequences from the subgroups 2b or 2c, respectively (with a variable order and/or number of repeat subtypes). Furthermore, in the groups "test 3," "test 4," and "test" the separation index and the percentage of correctly aligned repeat positions drops significantly in both reference methods.

As can be seen from Table I, the new alignment method aligns more repeat tuples correctly than the other two protocols when providing exact repeat boundaries. The gain of correctly aligned repeat positions rises from 0.5% ("test 2a") to >25% (group "test") of the correctly aligned repeat positions and comprises in total >300,000 tuples. Note that this number is bigger than the possible maximum of correctly aligned positions in other groups. By definition of the strict alignment mode, none of the declared repeat positions can be aligned with any positions outside of the repeats. Therefore, the number of misaligned SP remains zero in all instances, resulting in an SI index of 100%, which proves the principles of our method.

TABLE I. Comparison of the Novel Method (RAlign) in the Different Stringency Modes (*strict*, respectively *relaxed*) With Standard Alignment Methods (T-Coffee, Dialign)

	method	repeat-boundaries	mode	$SP_{correct}$		SP_{wrong}		SI [%]		t [sec]	
test 1a (5)	T-Coffee	none		8,790	(17,809)	656	(890)	79.39	(78.36)	347	
	Dialign	none		8,877	(18,011)	532	(708)	78.43	(77.71)	21	
	RAlign	noisy	strict	5,614	(10,586)	648	(866)	65.41	(64.56)	19	
	RAlign	noisy	relaxed	8,941	(16,780)	226	(446)	90.16	(88.51)	45	
	RAlign	exact	relaxed	9,250	(19,071)	230	(254)	91.27	(91.81)	157	
	RAlign	exact	strict	9,579	(19,465)	0	(0)	100.00	(100.00)	24	
test 1b (6)	T-Coffee	none		6,700	(11,672)	2,954	(3,794)	58.91	(57.49)	203	
	Dialign	none		7,475	(12,421)	782	(1,186)	86.62	(84.70)	10	
	RAlign	noisy	strict	1,994	(3,246)	4,402	(7,424)	35.67	(33.87)	12	
	RAlign	noisy	relaxed	6,935	(10,907)	1,188	(1,666)	82.17	(79.93)	26	
	RAlign	exact	relaxed	7,901	(14,134)	606	(712)	88.79	(89.14)	78	
	RAlign	exact	strict	9,170	(15,817)	0	(0)	100.00	(100.00)	8	
test 2a (12)	T-Coffee	none		429,730	(467,264)	578	(600)	99.69	(99.68)	1,974	(1)
	Dialign	none		430,247	(467,848)	124	(134)	99.81	(99.80)	140	
	RAlign	noisy	strict	420,859	(455,382)	254	(276)	95.00	(96.79)	1,677	
	RAlign	noisy	relaxed	424,762	(462,314)	228	(254)	99.46	(99.41)	2,468	
	RAlign	exact	relaxed	432,297	(470,745)	50	(60)	99.84	(99.82)	7,056	
	RAlign	exact	strict	432,402	(470,873)	0	(0)	100.00	(100.00)	1,222	
test 2b (23)	T-Coffee	none		123,969	(139,088)	8,276	(11,216)	86.45	(86.22)	1,868	
	Dialign	none		134,660	(150,677)	1,906	(2,746)	93.26	(93.25)	49	
	RAlign	noisy	strict	57,008	62,634	10,846	(13,472)	62.19	(61.47)	694	
	RAlign	noisy	relaxed	114,590	(128,221)	3,608	(4.972)	89.77	(89.51)	274	
	RAlign	exact	relaxed	139,003	(156,342)	1,024	(1,580)	97.15	(97.08)	900	
	RAlign	exact	strict	140,573	(158,600)	0	(0)	100.00	(100.00)	494	
test 2c (25)	T-Coffee	none		43,967	(65,136)	31,212	(52,766)	58.19	(55.68)	1617	
	Dialign	none		58,661	(82,786)	10,946	(16,104)	81.60	(80.28)	76	
	RAlign	noisy	strict	17,822	(26,140)	21,992	(34,282)	43.69	(42.03)	661	
	RAlign	noisy	relaxed	47,666	(65,482)	10,206	(15,492)	81.50	(79.33)	539	
	RAlign	exact	relaxed	64,905	(101,552)	3,806	(5,454)	92.39	(92.05)	696	
	RAlign	exact	strict	70,835	(107,548)	0	(0)	100.00	(100.00)	946	
test 3 (9)	T-Coffee	none		180,110	(205,206)	25,456	(28,038)	62.42	(62.06)	3930	
	Dialign	none		193,787	(218,508)	6,094	(6,948)	88.38	(87.77)	117	
	RAlign	noisy	strict	105,162	(113,886)	27,668	(31,568)	61.31	(59.59)	32,472	(1)
	RAlign	noisy	relaxed	156,742	(177,588)	11,448	(12,422)	87.94	(86.76)	36,618	(1)
	RAlign	exact	relaxed	198,218	(225,106)	14,862	(15,410)	92.02	(91.74)	2,817	
	RAlign	exact	strict	204,282	230,877	0	(0)	100.00	(100.00)	1,556	
test 4 (6)	T-Coffee	none		11,674	(19,007)	4,020	(6,410)	64.49	(63.33)	830	
	Dialign	none		13,283	(21,069)	1,088	(1,578)	87.08	(86.75)	33	
	RAlign	noisy	strict	7,645	(12,625)	3,856	(5,194)	52.81	(51.27)	517	
	RAlign	noisy	relaxed	14,500	(21,873)	1,558	(2,146)	87.07	(85.27)	482	
	RAlign	exact	relaxed	12,606	(20,135)	90	(104)	99.32	(99.28)	298	
	RAlign	exact	strict	14,692	(23,432)	0	(0)	100.00	(100.00)	70	
test (13)	T-Coffee	none		1,351,044	(1,601,793)	447,470	(599,416)	69.87	(69.21)	93,671	(2)
	Dialign	none		1,480,414	(1,792,597)	314,126	(399,766)	86.85	(85.40)	1,889	
	RAlign	noisy	strict	948,711	(1,185,521)	560,606	(677,014)	75.27	(73.70)	85,970	(3)
	RAlign	noisy	relaxed	1,430,393	(1,715,455)	317,558	(400,828)	85.84	(84.35)	90,799	(3)
	RAlign	exact	relaxed	1,846,165	(2,328,443)	208,674	(251,220)	93.67	(92.57)	73,653	
	RAlign	exact	strict	1,903,708	(2,403,126)	0	(0)	100.00	(100.00)	38,524	
total (99)	T-Coffee	none		2,155,984	(2,526,975)	520,622	(703,130)	72.43	(71.50)	95,051	(3)
	Dialign	none		2,327,404	(2,763,917)	335,598	(429,170)	87.75	(86.96)	2,335	
	RAlign	noisy	strict	1,564,815	(1,870,020)	630,272	(770,096)	61.42	(60.41)	119,786	(4)
	RAlign	noisy	relaxed	2,204,529	(2,598,620)	346,020	(438,226)	87.99	(86.63)	128,738	(4)
	RAlign	exact	relaxed	2,710,345	(3,335,528)	229,342	(274,794)	92.75	(92.24)	85,655	
	RAlign	exact	strict	2,644,668	(3,271,138)	0	(0)	100.00	(100.00)	42,844	

For each RAlign run, repeat boundaries are provided once exactly as given by the reference (*exact,* shaded in dark gray), and another time with noise as often produced by automatic repeat detection (*noisy,* light gray). Results are presented for each of the groups (*test 1a-4*) and their summary (*total*), for each one the number of alignments comprised is given in parentheses. Each time the correctly aligned repeat tuples ($SP_{correct}$) and the misaligned tuples (SP_{wrong}) are shown for the core regions, and in parentheses for all of the repeat areas (i.e., core regions and positions with no superimposed structural information). The separation index (*SI*) of a group is derived as the arithmetic average of the *SI* indices over all included alignments and is given in percent points. Time complexity is given as the number of seconds a respective run required, followed by the number of failures within parentheses. Alignments that did not terminate within 24 h or that ran out of memory on a Pentium 4 (3.4 GHz, 2GB RAM), had to be diverted to a SunFire 880 (32GB RAM) for successful completion and are considered as *failures* (given in parentheses to the right).

When relaxing the constraints, sometimes anchors contradicting the repeat pattern are chosen by error during the greedy selection process due to their high sequence similarity. These wrongly selected anchors efficiently prevent from aligning many positions correctly. Nevertheless, even with relaxed constraints, the SI index of the repeat-sensitive alignment remains significantly beyond that of the two reference methods, as long as the repeat boundaries provided are correct (Table I).

Table I also shows the results yielded when applying repeat boundaries that are rather insecure. To evaluate our method with noisy constraints, we provide repeat boundaries for the BAliBase alignments as predicted by an automatic repeat detection method (Trust[10]) and compare the results yielded by the strict alignment mode with those achieved under relaxed constraints. To calculate the SP of correctly aligned and misaligned positions, the repeats as annotated in BAliBase are taken as reference. Here, the strict mode in all tested cases led to results below the quality of the reference aligners. The relaxed constraint is aimed to help the alignment in case repeat boundaries provided are unreliable. Table I clearly shows that under relaxed constraints, the number of correctly aligned repeat residues increases whereas that of misaligned repeat positions is reduced, resulting in a higher average SI for each of the test groups. In total, correctly aligned positions are increased by about 50% and also incorrect cross-alignments are approximately cut in half when applying the relaxed alignment mode compared with the strict alignment mode on noisy boundaries.

This significant gain is primarily due to the following fact: Trust, like many automatic repeat detection programs, does not provide any interface to process multiple-sequence sets. To be specific, repeats are detected by Trust for each sequence in single, unaware of the other sequences in the input set. Boundaries of repeats detected hence may vary substantially across the sequence set, resulting in a shorter intersecting substring. Within the strict alignment mode, these intersection "losses" caused by the boundary variations effectively prevent the selection of many correct anchors. Consequently, anchors contradicting the repeat pattern are chosen and cannot be taken back later on during the greedy selection process. Providing repeat boundaries with a strong boundary variation across the sequences, therefore, is worse than not providing repeat boundaries at all. However, if the quality of repeat boundaries is not known, the relaxed stringency mode can at least perform as well as the two reference methods as shown in Table I. Note that "performance" here only refers to the SI measure. It does not take into account the geometry of the aligned repeat units (e.g., there is no penalty for repeats that are torn into pieces to fit).

Regarding the time benchmarks, the algorithms can be split into different complexity groups: T-Coffee needs $O(k^3|s|^2)$ time, whereas Dialign has a worse time complexity of about $O(k^4|s|^2)$ with $|s|$ denoting the maximal sequence length in the input set. However, by default, Dialign disables the calculation of the sensitive overlap weights for large data sets (>35 sequences), such that the time complexity falls back to a maximum of $O(k^3)$ and in average it requires $O(k^2)$ to $O(k^3)$ time. The time complexity of the repeat-sensitive alignment method using the strict constraint depends on the underlying Dialign method running on the repeats-intervening sequence fragments, which is more time-efficient than aligning complete sequences. On the other hand, additional calculations have to be performed to remap and rescore the anchors found. On balance, the novel method usually takes more time than Dialign (Table I), but it can also perform slightly faster (Table I, "test 1b"), depending on the corresponding number and position of the repeats. In contrast, RAlign with relaxed constraints relies on two full Dialign runs: one on the intervening sequences and one on the complete sequences. With the additional all-against-all comparison of the repeat anchors that is necessary for the calculation of $olw(f)$, computation times are significantly higher than those of corresponding Dialign runs (Table I).

As expected, strict alignments are computed faster in most instances than the corresponding alignments under relaxed constraints (Table I, "total"). However, it is possible that the algorithm in relaxed mode reduces the number of anchors before the time-critical step of calculating the overlap weights. In such cases, it can perform faster in relaxed mode than under strict constraints (e.g., Table I, "test 2c"). For computation time for alignments with noisy boundaries, it should be taken into account that an exact global alignment of the repeats is calculated (divide-and-conquer alignment[12]). Therefore, input sets with many repeats that are not supporting the alignment very well can take a long time to align (Table I, "test").

Case Studies

Although the benchmark tests shown in the previous section show some benefits of the new method, the SI indices presented suffer from a lack of detail. First, they do not include any measurement about the quality of the non-repetitious sequence areas that have been aligned. As mentioned earlier, no reference alignments exist for the intervening regions and there are also no unique markers annotated, which could be used as a reference. However, this only slightly affects the results because most of the BAliBase test sets do not have extensive non-repetitious sequence information (i.e., most sequence stretches consist of tandem repeats). More severe is the fact that the sum of correctly aligned repeat tuples (SP) does not consider whether the tuples involve positions of the same repeat unit. Therefore, especially in sequences with a lower number of repeat copies, it can easily happen with traditional methods that repeat sequences are aligned with segments of more than one repeat in another sequence. Such repeats are torn apart, and the substrings of their sequence are scattered in the alignment. To investigate in more detail the quality of the motif-constrained alignment method, we present alignments of two protein families.

Src homology domains SH3 and SH2

We selected from the BAliBase reference sets 15 sequences with multiple copies of the Src homology domain 3

(SH3) and thus composed a sequence set with sequences containing a variable number of SH3 copies; most sequences have two, but some sequences contain three SH3 repeats. The SH3 module is a small protein domain of about 50 amino acid residues first identified in the noncatalytic part of several cytoplasmic protein tyrosine kinases (e.g., Src). Since then, it has been found in a great variety of other intracellular or membrane-associated proteins, across many organisms (e.g., *Homo sapiens, Mus musculus, Gallus gallus, Drosophila melanogaster, Caenorhabditis elegans, Saccharomyces cerevisiae,* etc.). Repeat boundaries for the SH3 motifs were taken as specified by the BAliBase reference.

In addition, all selected proteins contained a single conserved Src homology 2 (SH2) domain. The SH2 module is widely spread over proteins that act as regulatory modules involved in intracellular signaling cascades. SH2 domains contain about 100 amino acids and hence are twice as long as SH3 domain sequences. SH2 and SH3 motifs do not share any significant mutual sequence similarity. As a test, we did not provide the boundaries of the SH2 motifs as input to our algorithm, because the motif is not repeated in any of the individual sequences and, therefore, is treated as a non-repeat fragment. The SH2 domains instead are used as markers to assess the performance of our method.

Figure 7 provides a schematic outline of the alignments obtained for the data set when applying T-Coffee [Fig. 7(a)], Dialign [Fig. 7(b)], and the repeat-sensitive alignment method [Fig. 7(c)] to the input. Although T-Coffee generally aligns the SH3 domains accurately, it fails to delineate the relation between the SH2 domains [Fig. 7(a)]. In contrast, the Dialign method matches the SH2 domains correctly, but seriously scatters the SH3 domains [Fig. 7(b)]: the N-terminal two SH3 domains in the Dreadlocks protein ("DOCK_DROME," Appendix B) remain unaligned with counterparts in the other sequences, whereas an aligned block of N-terminal SH3 domains in the 11 top sequences shows an incorrect sequential overhang over three SH3 domains corresponding to the N-terminal two SH3 domains in the bottom three sequences [Fig. 7(b)]. Moreover, a 48-residue homologous region within intervening regions of the Nck family remains unmatched as well. When applying our repeat-sensitive alignment, the SH2 and SH3 domains are all aligned biologically correctly [Fig. 7(c)].

Apolipoproteins

Apolipoproteins are lipoprotein particles that make hydrophobic lipids solvable in an aqueous environment and transport them via the blood to certain target cells. To do so, their core is formed by particles of hydrophobic lipids, surrounded by the apolipoproteins with amphiphilic helices that provide a polar, water-solvable surface. These helices are encoded by exon 4 of the respective gene, and most of them are made up of two 11-mers; however, some only contain a single 11-mer. Thus, many of the repeats bear some intrinsic symmetry, an attribute that is not unusual for repeats. Repeat copies are tandem-repeated within the sequence, and their number varies substantially across the 10 different apolipoproteins yet isolated (i.e., apolipoprotein A-1, A-2, A-4, B-48, B-100, C-1, C-2, C-3, D, and E). The strongly varying copy number, the intrinsic symmetry, and the low similarity between some of the repeats make the sequences rather hard to align. Moreover, the helix-structure forces residues neighboring within the linear sequence to opposite sites of the helix. Hence, an alignment of the repeats with exactly matched boundaries is required (Appendix B; see Fig. 11).

We selected seven in-depth investigated sequences[15] and submitted the repeat boundaries of the 22mer repeats together with the input sequences to the repeat-sensitive alignment method (strict constraint mode). Both reference alignment methods, Dialign and T-Coffee, have different copy numbers, poor conservation, and internal symmetry of the repeats [Fig. 8(a–b)]. Although the repeats are aligned accurately in the highly conserved apolipoproteins A-1 (APA1_HUMAN and APA1_RAT), the less similar copies of other sequences are clearly shifted in the alignments. In contrast, when constraining the alignment to the boundaries of the 22mer motifs, we get a much clearer impression of the relationship in between the repeats [Fig. 8(c)]. From the repeat-aware alignment, the closest relations according to sequence similarity along the repeats become apparent. Further biological investigation will be necessary to discover whether the aligned repeat copies are indeed orthologous and whether the unaligned repeat copies did evolve by duplication (paralogs).

DISCUSSION

The novel method (RAlign) is, to our knowledge, the first automated strategy to perform global multiple alignment while keeping track of repeats within the input sequences. It is based on the anchoring procedure that has been introduced by the Dialign method.[5] Time benchmarks showed the new method to perform 1–2 orders of magnitude slower than the original Dialign protocol (Table I). In the process of publishing this work, Morgenstern et al. developed a user-constrained version of Dialign that also was applied to repeats.[16] However, because anchors and weights are to be set manually by a biological expert, this method focuses on enhancing single alignments rather than providing a general solution for repeated motifs. Recently, another method for nonlinear multiple alignment of shuffled and repeated elements was proposed, wherein local similarities in the input sequences are represented as a (possibly cyclic) graph.[17]

The dilemma is that sometimes the repeats are obstacles in retrieving relevant information of a sequence set, whereas in other instances, they might serve as anchors to guide the alignment. We solved this by a separated evaluation of the information contained in both parts (i.e., the repeated areas and the intervening sequences) and merging modes of different stringency for both pieces of information (i.e., the strict and the relaxed alignment mode). If the boundaries of the submitted motifs are reliable, the user should select strict alignment constraints, whereas in noisy information, the relaxed align-

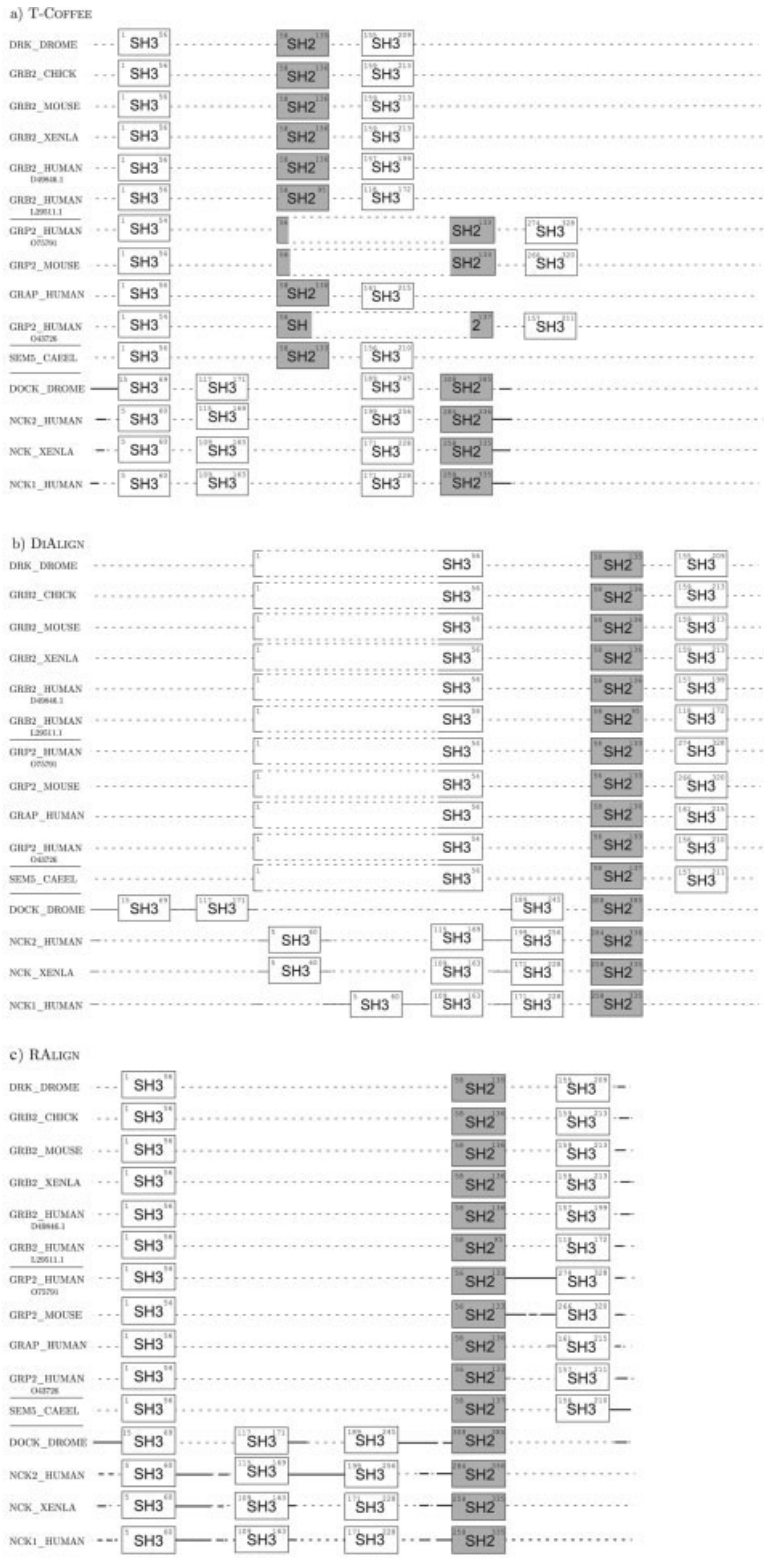


Figure 7. Schematic multiple alignment generated by (**a**) T-Coffee, (**b**) Dialign, and (**c**) the repeat-sensitive alignment (RAlign) when applied to a set of protein sequences containing a varying number of SH3 copies (transparent boxes). Note that the boundaries of the additional SH2-domains (boxes shaded in gray) were not provided with the input. In each of the repeat boxes, the delimiting N- and C-terminal residue numbers are given.

Figure 8. Alignment of seven of the main apolipoproteins in human. The repeats consist of 11-mers (boxes labeled S, single repeats), which mostly are joined to 22 amino acids long units (22-mers, double repeats). An 11-mer is shaded in light green if they correspond to the first half, dark green if they correspond to the second half of a double repeat according to Luo and Li.[15] Because of the intrinsic symmetry and the bad conservation of the repeats during evolution, T-Coffee (**a**) and Dialign (**b**) cannot match them correctly. In contrast, the repeat-sensitive alignment method (**c**) retrieves an alignment where all repeat boundaries are matched correctly.

ment is more likely to improve the alignment results. Strong noise caused by inaccurate repeats information, however, cannot always be handled well by our new method, so that it is possible that our algorithm—even when run in relaxed mode—is unable to outperform traditional MSA methods. Nonetheless, when reliable repeat boundaries are provided, our method produces significantly improved alignments (Table I).

It would be highly interesting to know, which threshold of noise finally brings down the objective function, and we suggest to perform tests with different repeat detection methods or randomized repeat boundaries in the future. However, additional effort will be necessary to complete this task, because a measure for the repeat noise has to be set up that efficiently captures attributes affecting the greedy selection of the anchors.

Furthermore, we see a potential of the method when investigating repeat histories: in strict alignments, repeats with a relation in the global context are aligned, whereas the ones that cannot be matched remain unaligned so that positions of the repeats are no longer misaligned with positions of the intervening sequences (Table I). These alignment layouts may serve as additional information to analyze repeat duplication histories, or to study the phylogenetic relationship of repeat clusters. Finally, the method is rather general and can be applied not only to repeats but also to motifs based on another criterion (e.g., non-repetitious motifs, structural units, etc.).

ACKNOWLEDGMENTS

The work was supported by a scholarship of the Ernst Schering Research Foundation to MS who is also grateful

to the staff of the Vrije Universiteit for the hospitality. Further thanks go to Jens Stoye and Burkhard Morgenstern for fruitful discussions.

REFERENCES

1. Benson G. Sequence alignment with tandem duplication. J Comp Biol 1997;4:351–367.
2. Sammeth M, Weniger T, Harmsen D, Stoye J. Alignment of tandem repeats with excision, duplication, substitution and indels (edsi). LNBI (Proceedings of WABI '05) 2005;3692:276–290.
3. Notredame C. Recent progress in multiple sequence alignment: a survey. Pharmacogenomics 2002;3:131–144.
4. Notredame C, Higgins D, Heringa J. T-COFFEE: a novel method for fast and accurate multiple sequence alignment. J Mol Biol 2000;302:205–217.
5. Morgenstern B. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. Bioinformatics 1999;15:211–218.
6. Thompson JD, Plewniak F, Poch O. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. Bioinformatics 1999;15:87–88.
7. Kurtz S, Schleiermacher C. REPuter: fast computation of maximal repeats in complete genomes. Bioinformatics 1999;15:426–427.
8. Heringa J, Argos P. Rapid automatic detection and alignment of repeats in protein sequences. Proteins 2000;41:224–237.
9. Heger J, Holm L. A method to recognise distant repeats in protein sequences. Proteins 1993;17:391–411.
10. Szklarczyk R, Heringa J. Tracking repeats using significance and transitivity. Bioinformatics 2004;20 (Suppl 1):I311–I317.
11. Tönges U, Perrey S, Stoye J, Dress A. A general method for fast multiple sequence alignment. Gene 1996;172:GC33–GC41.
12. Stoye J. Multiple sequence alignment with the divide-and-conquer method. Gene 1998;211:GC45–GC56.
13. Sammeth M, Morgenstern B, Stoye J. Divide-and-conquer alignment with segment-based constraints. Bioinformatics 2003;19: ii189–ii195.
14. Burset M, Guigó R. Evaluation of gene structure prediction programs. Genomics 1996;34:353–357.
15. Luo CC, Li WH. Structure and evolution of the apolipoprotein multigene family. J Mol Biol 1986;187:325–340.
16. Morgenstern B, Prohaska SJ, Werner N, Weyer-Menkhoff J, Schneider I, Subramanian AR, Stadler PF. Multiple sequence alignment with user-defined constraints. In German Conference on Bioinformatics, Lecture Notes in Informatics, 2004.
17. Raphael B, Zhi D, Tang H, Pevzner P. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, 14:2336–2346, 2004.

APPENDIX A: THE DIALIGN ALGORITHM

Dialign[5] is an anchor-based approach to produce global alignments. Here, anchors (i.e., gap-free pairwise local alignments) from a multiple-sequence set are iteratively collected and greedily added to the solution. Parts of the method that are relevant to understand this work are briefly described in the following sections.

The Objective Function

The objective function $w(f)$ assigns *weights* to each anchor $f = ([p,i],[q,j],\ell)$ based on the probability $P(\ell_f,\sigma_f)$ that the corresponding subsequences are not related, that is, the probability that an alignment of two random strings with length ℓ_f have at least the similarity of σ according to some substitution table (e.g., Blosum62 for amino acid sequences). Note that σ_f is corrected by a factor dependent on the lengths $|s_p|$ and $|s_q|$ of the involved sequences. Finally, motivated by traditions established in statistical mechanics and information theory, the negative logarithm of this error probability is considered:

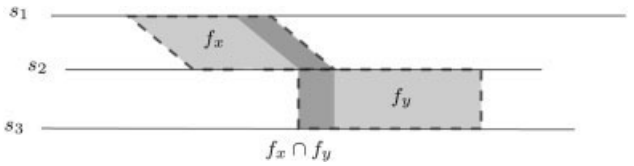


Figure 9. Two overlapping anchors f_x and f_y (gray links). Their weights $w(f_x)$ [respectively $w(f_y)$] are increased by the weight of the anchor $f_x \cap f_y$ induced by the overlapping area (dark shadowed) to calculate the overlapping weights $olw(f_x)$ [and correspondingly $olw(f_y)$].

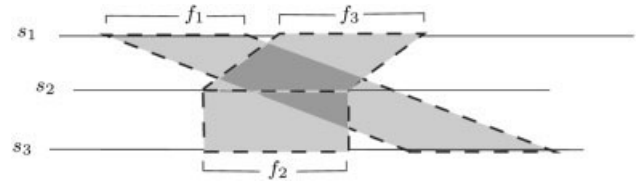


Figure 10. Transitive links and inconsistencies in the common sequence context: although all of the anchors f_1, f_2, and f_3 are consistent within the context of the respective sequence pairs, they are not realizable at the same time. The transitive link between s_1 and s_3 through the anchors f_2 and f_3 is contradicting the relationship imposed by f_1.

$$w(f) = -\ln P(\ell_f,\sigma_f)$$

To increase the sensitivity for less conserved motifs, a secondary scoring function has been defined. It enlarges the $w(f)$ by the weight of regions overlapping with other anchors. Note that for calculating this so-called overlapping weight olw_F, only the set of pairwise consistent anchors of the longest diagonal is taken into account. See Morgenstern[5] for a more detailed definition of weights and overlap weights, a draft overview is given in Figure 9.

$$olw_{(fx)} = w_{fx} + \sum_{fy \neq fx} w(f_x \cap f_y)\ \forall f_y \in \tilde{F}$$

Consistency and Transitivity

Global multiple alignment by anchor assembly has to deal with two sources of inconsistency (i.e., anchors that are not realizable in a common global alignment). Pairwise consistency requires the anchors found for each pair of sequences (s_p,s_q) from the input set $S=\{s_1,s_2,\ldots,s_k\}$ to be consistent which each other, but not necessarily with anchors found for other pairs of sequences from S. The set of pairwise consistent anchors is denoted by $\tilde{F}$, which is crucial for the calculation of correct overlap weights $olw_F(f)$. Pairwise inconsistent anchors can disproportionately bias regions with multiple possibilities for alignment (Fig. 6).

In contrast, consistency within the entire multiple-sequence context means that all anchors have to be consistent. The latter is more restrictive because transitive links of anchors sharing a common area can align positions that are not directly joined by an anchor. We denote with F a set of anchors consistent regarding all of the multiple sequences S, and the Dialign algorithm tries to find a high-scoring set F by greedily adding consistent anchors according to their overlap weight (Fig. 10).

APPENDIX B: SEQUENCE DETAILS
Accession Numbers

TABLE II. Swiss-Prot Accession Numbers of the SH3 Proteins

Protein	Organism	Sequence name	Accession number
Dreadlocks	*Dros. melanog.*	DOCK_DROME	Q24218
Protein E(sev)2B	*Dros. melanog.*	DRK_DROME	Q08012
Growth factor receptor-bound protein 2	*Gallus gallus*	GRB2_CHICK	Q07883
	Xenopus laevis	GRB2_XENLA	P87379
	Mus musculus	GRB2_MOUSE	Q60631
	Homo sapiens	GRB2_HUMAN	(P29354) D49846.1
	Rattus norveg.	GRB2_HUMAN	(P29354) L29511.1
SH2/SH3 adaptor protein NCK	*Xenopus laevis*	NCK_XENLA	P79956
Cytoplasmic protein NCK1	*Homo sapiens*	NCK1_HUMAN	P16333
Cytoplasmic protein NCK2	*Homo sapiens*	NCK2_HUMAN	O43639
Mona (GRB2-related adaptor protein)	*Homo sapiens*	GRP2_HUMAN	O75791 O43726
	Mus musculus	GRP2_MOUSE	O89100
Grap (GRB2-related adaptor adaptor)	*Homo sapiens*	GRAP_HUMAN	Q13588
Sex muscle abnormal protein 5	*Caenorh. elegans*	SEB5_CAEEL	P29355
Intersectins	*Homo sapiens*	ITN1_HUMAN	Q9UNK1
	Homo sapiens	ITSN_HUMAN	Q15811
	Homo sapiens	ITN2_HUMAN	O95062

TABLE III. Swiss-Prot Accession Numbers of the Apolipoproteins

Protein	Organism	Sequence name	Accession number
Apolipoprotein A-I [Precursor]	*Homo sapiens*	APA1_HUMAN	P02647
Apolipoprotein A-I [Precursor]	*Rattus norvegicus*	APA1_RAT	P04639
Apolipoprotein A-II [Precursor]	*Homo sapiens*	APA2_HUMAN	P02652
Apolipoprotein C-I [Precursor]	*Homo sapiens*	APC1_HUMAN	P02654
Apolipoprotein C-II [Precursor]	*Homo sapiens*	APC2_HUMAN	P02655
Apolipoprotein C-III [Precursor]	*Homo sapiens*	APC3_HUMAN	P02656
Apolipoprotein E [Precursor]	*Homo sapiens*	APE_HUMAN	P02649

Apolipoprotein Structure (Fig. 11)

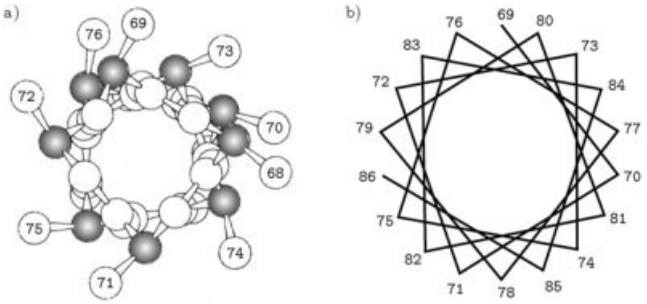


Figure 11. The 3D structure of apolipoproteins forces neighboring positions of the linear sequence to opposite sides of an alpha-helix. **a:** space-filling model of the first 8 residues and (**b**) schematic view of the complete helix formed by the first 22mer-repeat of human apolipoprotein A1.

Comparing Tandem Repeats assuming Duplications and Excisions of Variable Degree

Michael Sammeth and Jens Stoye

***Abstract*— Traditional sequence comparison by alignment employs a mutation model comprising two events, substitutions and indels (insertions or deletions) of single positions. However, modern genetic analysis knows a variety of more complex mutation events (e.g., duplications, excisions and rearrangements), especially regarding DNA. With the ever more DNA sequence data becoming available, the need to accurately compare sequences which have clearly undergone more complicated types of mutational processes is becoming critical.**

Herein we introduce a new method for pairwise alignment and comparison of sequences with respect to the special evolution of tandem repeats: substitutions and indels of single positions, and additionally duplications and excisions of variable degree (i.e., of one or more repeat copies simultaneously) are taken into account. To evaluate our method, we apply it to the *spa* VNTR (variable number of tandem repeats) cluster of *Staphylococcus aureus*, a bacterium of high medical importance.

Index Terms*— *sequence comparison, tandem repeats, variable number of tandem repeats, repeat duplication history, pairwise alignment, dynamic programming.

I. Introduction

Tandem repeats are a big challenge in sequence comparison. Traditionally, for the comparison of two sequences, so-called *edit operations* have been defined which represent the atomic steps of evolution. By successively applying such edit operations, the compared sequences can be aligned to each other and – assuming parsimony as a major characteristic of evolution – good sequence alignments minimize the number of operations for these conversions or, more precisely, the assigned *costs*. In the classical model of mutation, two types of edit operations are considered: the *substitution* and the *insertion* or *deletion* (together *indel*) of single characters in a sequence.

Whereas this model has proven to work well in general sequence alignments, modern genetics knows more complex sources of mutation – especially when regarding the evolution of DNA. These mechanisms affect no longer only single positions but complete substrings of a sequence. Common such edit operations are *duplications* (insertions of copied substrings; in case of *tandem* duplications, immediately adjacent to the original), *excisions* (deletions of substrings of a sequence), and *rearrangements* (relocations or reorientations of substrings within the sequence, e.g. *transpositions* or *inversions*).

Manuscript received February 15, 2006; revised ..., 2006. This work was supported by a doctoral fellowship of the Ernst Schering Research Foundation and a post-doctoral fellowship of the German Academic Exchange Service to MS.

M. Sammeth, and J. Stoye are with the Faculty of Technology, Bielefeld University, Germany.

Variable Number of Tandem Repeats (VNTR) loci provide a source of very informative markers in bacteria. Tandem repeats for bacterial identification have proven their utility for the typing of highly monomorphic pathogens such as *Bacillus anthracis*, *Yersinia pestis* [1], *Mycobacterium tuberculosis* [2], [3], or *Staphylococcus aureus* [4], [5]. *S. aureus* for instance, a bacterium responsible for a wide range of human diseases (e.g., endocarditis, toxic shock syndrome, skin, soft tissue and bone infections, etc. [6]), contains polymorphic 24-bp VNTRs in the 3' coding region of the staphylococcal protein A (the *spa* protein) [7]. The tandem repeats in this region undergo a mutational process including the duplication and excision of repeat copies in addition to nucleotide-based substitutions and indels [5], probably caused by slipped strand mispairing [8]. Further on, the microvariation of the *spa* VNTR cluster [4] seems to support the phylogenetic signal reported by other methods (e.g., by [9]). Therefore, an automated method to compare strains of *S. aureus* and classify them according to the microvariation of the *spa* tandem repeats is critical in order to determine the types of newly acquired sequences rapidly and accurately.

In recent years quite some work has been invested in the algorithmic study of tandem repeats. Tandem duplications and excisions follow different rules than regular, character-based indels. On the one hand the inserted or deleted substrings are usually much bigger in duplications and excisions, and on the other hand they contain the pattern of the tandem repeats in the corresponding sequence. Previous work in this field can roughly be categorized into (1) tandem repeat *detection*, (2) *alignment* of sequences containing tandem repeats (with or without knowledge of their positions), and (3) reconstruction of a tandem repeat *history* where the phylogenetic history of the tandem repeats of one sequence is tracked down to a single ancestor repeat. (1) concerns the detection of tandem repeat copies with an unknown pattern, e.g. in [10], [11]. In the context of (2), various works extended the model of nucleotide mutation to additionally consider tandem duplication events [12] and excisions of single tandem repeats (*degree 1*) [13], [14]. The research of (3) investigates possible duplication histories of the tandem repeats in a sequence in order to find one with minimal costs (*minimum tandem repeat history problem*), see [15], [16], [17], [18].

In this paper we extend methods developed in the context of category (2), but we also address aspects of category (3). We introduce an extended model of evolution where the duplication and excision of one or more consecutive tandem repeats (*variable degree* ≥ 1) is considered. Moreover, these operations may occur arbitrarily cascaded with mutations and

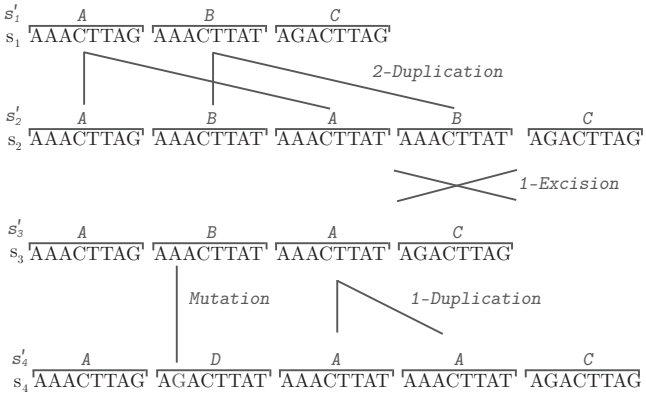


Fig. 1. An example for cascaded duplication, excision, and mutation events. Shown are DNA sequences s_i and the corresponding sequences s'_i on the macro alphabet Σ' of repeat types (superimposed on s_i in grey). The described edit operations successively are performed on the sequence. It can be seen easily that after a couple of cascaded operations the original sequence of characters is rather scrambled.

indels of single nucleotides. In Section II we formalize the evolutionary model and give an overview of the problem addressed. In Section III we introduce the structure of *contramers* (contraction units) to formally describe duplication events in the model. Using contramers, we propose in Section IV an exact algorithm to align and compare a pair of sequences. Finally, in Section V, we give some practical examples for comparing *spa* sequences of *S. aureus* with our method, and in Section VI we discuss the benefit of the evolutionary model and its potential for accurate investigations.

II. The Evolution of *spa* Types

Let s be a sequence of characters over the DNA alphabet $\Sigma = \{\text{A,C,G,T}\}$, and let s consist of tandem repeats. If the boundaries of the repeats are known, s can be written directly as a sequence s' over the macro alphabet of the different repeat types $\Sigma' = \{-, A, B, C, D, \ldots\}$. The additional gap character ($- \in \Sigma'$) is used later on when aligning repeat sequences (Section IV). A sequence $s' \in (\Sigma')^+$ is called a *VNTR sequence*, where $(\Sigma')^+$ denotes the set of all nonempty strings over Σ'. On VNTR sequences we define an evolutionary model, allowing duplications and excisions of repeats (characters in s'), as well as substitutions and indels of nucleotides within the repeats (Fig. 1). We call this the *VNTR model*[1].

Since the commonly used substitutions and indels work on the DNA bases of s, we comprise them in a bundled *mutation* operation $mut(a, b)$, which changes a tandem repeat $a \in \Sigma'$ to a tandem repeat $b \neq a$. In contrast, the duplication and excision events always affect complete repeats of s'. Precisely, a duplication event $dup(w)$ occurring during VNTR evolution can be of *variable degree*, replacing a substring w of s' by two concatenated copies ww. Since w may contain one or more tandem repeat(s), the corresponding duplication is sometimes named according to $|w|$: 1-duplication, 2-duplication, etc. [12], or duplication of order $|w|$ [20]. In this paper we use the notation from [12], not to get confused with the evolutionary *order* of edit operations. In the same manner, the excision of a substring w of s', denoted $exc(w)$, can be of variable degree: 1-excision, 2-excision, etc.

Herein we describe a model of duplication/excision events at certain positions in the nucleotide sequence s (*fixed boundaries*). However, duplicated/excised substrings may start and end anywhere in s' (*variable degree*). Furthermore, the duplication/excision operation in the model is *single-step* [12], denoting that no more than one copy of a duplicated/excised substring is produced in one evolutionary step (*arity* $= 1$ according to [20]). The *order* of events in evolution is unrestricted, i.e., all four edit operations described by the model may be applied arbitrarily cascaded with each other (Fig. 1).

In order to assess the evolutionary distance between two given sequences, we assign *costs* to all operations comprised in the evolutionary model: $cost(exc(w))$ and $cost(dup(w))$ for the excision respectively the duplication of the tandem repeats in string w, and $cost(mut(a, b))$ for a mutation of a repeat type a into the repeat type b. The *cost model* of VNTR evolution can then be freely adjusted[2] with respect to the following restrictions:

- Excision costs should be positive, $cost(exc(w)) > 0$ for all words $w \in (\Sigma')^+$, since excision events can replace all other operations. To be specific, any pair of sequences (s', t') can be derived from a concatenated ancestor string $s't'$ by two excisions: once excising t' yielding s' and once excising s' yielding t'. Hence, finding the minimum distance for sequences in a cost model with $cost(exc(w)) = 0$ is trivial. Furthermore, excision costs should fulfill the triangle inequality $cost(exc(w)) \leq cost(exc(w_1)) + cost(exc(w_2))$ whenever $w = w_1 w_2$.
- Duplication costs should be non-negative, $cost(dup(w)) \geq 0$ for all words $w \in (\Sigma')^+$.
- Mutation costs should comply with the properties of a metric: symmetry ($cost(mut(a, b)) = cost(mut(b, a))$ for all $a, b \in \Sigma' \setminus \{-\}$), zero property ($cost(mut(a, b)) = 0$ if and only if $a = b$ for all $a, b \in \Sigma' \setminus \{-\}$), and the triangle inequality ($cost(mut(a, b)) + cost(mut(b, c)) \geq cost(mut(a, c))$ for all $a, b, c \in \Sigma' \setminus \{-\}$).

The problem of sequence evolution comprising the described operations can now be formulated as an optimization problem with the goal of cost minimization.

Definition 1 (VNTR Distance): Given two VNTR sequences s' and t' and cost measures $cost(exc(w))$, $cost(dup(w))$, and $cost(mut(a, b))$, find the minimum sum of costs among all series of operations possible to reproduce s' and t' from a common ancestor string. This cost is called the *VNTR distance* $d(s', t')$ of s' and t'.

Definition 1 can be interpreted such that an (*a priori* unknown) common ancestor u' is subjected to evolutionary operations transforming it into s' and t', respectively. This series of operations represents a *VNTR transformation* $T^{(u')}_{s',t'}$ whose cost $cost(T^{(u')}_{s',t'})$ is the sum of the costs for the oper-

[1]Previously [19] we used the term EDSI (excision, deletion, substitution, indel) model, but since our duplications and excisions are restricted to whole repeat copies, we now prefer the term VNTR model.

[2]For a definition of the costs used for the *Staphylococcus aureus* evolution, see Section V.

ations it contains, and the transformation with the least costs defines the *VNTR distance* $d(s',t') = \min_{u'}(cost(T^{(u')}_{s',t'}))$.

Definition 2 (Optimality): A VNTR transformation $T^{(u')}_{s',t'}$ whose cost equals the VNTR distance between s' and t' is *optimal*. Transformations whose costs are larger than $d(s',t')$ are *suboptimal*.

To find an optimal transformation $T^{(u')}_{s',t'}$, one naively may try to enumerate all possible ancestors of s' in a set S' and all possible ancestors of t' in a set T', and then find a common sequence $u' \in S' \cap T'$ with $d(s',t') = \min_{u'}(cost(T^{(u')}_{s',t'}))$. However, the unrestricted order of excisions with respect to the other edit operations under the VNTR evolution implies an infinite search space of possible ancestor sequences. To be specific, when reconstructing possible evolutionary histories from a given pair of VNTR sequences (s',t'), theoretically there could have been present an arbitrarily large number of repeats between any two adjacent positions i and $i+1$ of s' (or correspondingly between adjacent positions j and $j+1$ of t'). Therefore, we limit our search to optimal transformations between s' and t'. With respect to optimality, the following limitations apply to operations preceding an excision event $exc(v)$ in s':

- Mutation events $mut(a,b)$ replacing a repeat type a by another repeat type b that succeedingly is excised by $exc(v)$ containing $b = v[x]$ for some x, $1 \leq x \leq |v|$, are suboptimal and can be substituted by the excision of v containing the original a without increasing the cost. Note that the argument also holds for multiple mutations on the same position of v and for the mutation of multiple positions in v.
- The excision of a substring w contained in or adjacent to v can be merged into a single excision of v and w without increasing the cost.
- If the excision of v is preceded by a duplication $dup(w)$ such that v overlaps the duplication result ww, three cases can be distinguished. Let $\bar{v}$ be the part of v overlapping ww: (a) if $|\bar{v}| > |w|$ or $|v| > |\bar{v}| = |w|$, then the duplication followed by the excision can be substituted by one excision without increasing the cost; (b) if $|v| = |\bar{v}| = |w|$, then $dupl(w)$ and $exc(v)$ are compensating events and will never be part of an optimal transformation; (c) if $|\bar{v}| < |w|$, then possible substitutions in an optimal transformation depend on the number of excisions and their location inside ww. In some instances no substitutes of equal or lower cost can be specified.

From these restrictions one can directly deduce the following.

Observation 1: In an optimal transformation, an excision $exc(v)$ may exclusively be preceded by a duplication $dup(w)$ such that v and the duplication result ww overlap and for the operlapping part $\bar{v}$ holds $|\bar{v}| < |w|$.

In consequence of Observation 1, the space of evolutionary operations to be explored in order to find the minimal distance $d(s',t')$ is finite.

Theorem 1 (finiteness): When comparing two VNTR sequences s' and t' under the VNTR model of evolution, the number of excised substrings v that lead to optimal transformations $\min_{u'}(cost(T^{(u')}_{s',t'}))$ is limited. The sets S' of ancestors with optimal transformations into s' and T' of ancestors with optimal transformations into t' are therefore finite.

Proof: Assume a non-empty substring v between positions i and $i+1$ on s' which has been excised by $exc(v)$. By Observation 1, in an optimal transformation any part $v[x_1,x_2]$ can have been produced either by an immediately preceding duplication $dup(w)$ overlapping v such that the overlapping part $\bar{v}$ is shorter than w or there have been no preceding operations on the positions.

In the first case, the $|w|-|\bar{v}|$ deleted repeats are identical to their duplicates in the other copy of w. Hence, reconstruction of the excised substring $\bar{v}$ is possible from the substrings in s' to the left of position $i+1$ and to the right of position i whose number is finite. The part of v not overlapping with ww may be overlapping with another duplication or is not preceded by any operation.

In the second case, $exc(v)$ is not preceded by any operation. Since $cost(exc(v)) > 0$, the excision operation can only contribute to $d(s',t')$ if there is an identical substring v (still) present in t': if t' does not contain any corresponding substring, $exc(v)$ can be eliminated to minimize the costs of the transformation; whereas, if t' contains a substring $\bar{v} \neq v$, additional mutation costs can be avoided by replacing $exc(v)$ by $exc(\bar{v})$. Thus, in this case the possibilities for insertions are limited by the substrings found in t'. ■

In the remainder of this paper, we describe an exact algorithm to compare and align VNTR sequences with respect to the defined edit operations $exc(w)$, $dup(w)$ and $mut(a,b)$. Basically the algorithm works in two steps: first it finds possible ancestors for all substrings of the VNTR sequences s' and t' separately (Section III), and then it determines the VNTR distance between s' and t' in a generalized sequence alignment procedure using the ancestors found before as alternative alignment possibilities between the compared sequences (Section IV).

While the time and space complexity of our algorithm are exponential w.r.t. the sequence lengths $|s'|$ and $|t'|$, note that the input of the algorithm are sequences of already annotated repeats and the input size therefore is much shorter than the original DNA sequences.

III. The Concept of Contramers

Although not observed in nature, we use the term *contraction* for the mathematically inverse process of a duplication. The first step of the algorithm is based on representing contraction units that we call *contramers*.

Definition 3 (contramer): Let s' be a VNTR sequence, b, m and e integers such that $1 \leq b < m \leq e \leq |s'|$, and A an alignment of the *prefix* $s'[b,m-1]$ and the *suffix* $s'[m,e]$. $C = (s',b,m,e,A)$ is a *contramer*, describing a contraction on the substring $s'[b,e]$ of s', split by the *meridian* m. The alignment A describes how the characters of both segments are evolutionarily related according to the contramer. To be specific, aligned repeats correspond to each other with respect to possible mutation events (*links*) and gaps indicate the excision of repeats.

Obviously contramers can capture single-step contractions reversing duplications of variable degree on s', as required

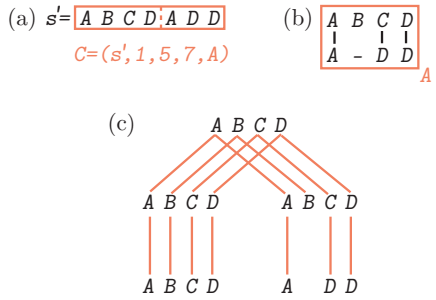


Fig. 2. (a) A contramer $C = (s', 1, 5, 7, A)$ that implies the duplication of substring $s'[1, 4] = ABCD$ and its post-duplicational modification into ADD. (b) The alignment shows that repeat B was excised while repeat C mutated to repeat D. All vertically adjacent repeat pairs (i.e., non-gap characters linked by black lines) in an alignment layout correspond to each other w.r.t. possible mutations. (c) A duplication-tree representation of these links.

by the model. Note that the duplicated repeat copies may be approximate, i.e. they may include mutations or excisions. However, contramers with one empty segment are not considered, due to the restrictions on optimal transformations described in Section II. An example of a contramer representing a duplication with subsequent excision and mutation is given in Fig. 2.

A. Primary library of contramers

The initial set of contramers is extracted directly from the repeat sequence s'. For each meridian position in s', $1 < m \leq |s'|$, all possible alignments of available non-empty prefixes $s'[b, m-1]$, $1 \leq b < m$, and non-empty suffixes $s'[m, e]$, $m \leq e \leq |s'|$, are generated. The contramers inferred thereby form the *primary library*. Note that at this stage links are generated exhaustively for every set of boundaries (b, m, e) and similarity of the aligned segments is not optimized by any objective function. This is necessary because later on (Section III-B) transitive links between overlapping contramers will be established and suboptimal alignments can be involved in optimal transformations.

Algorithm 1 (Generate contramers for the primary library L)

```
1:  L ← ∅
2:  for m ← 2 to |s'| do
3:    for b ← 1 to (m − 1) do
4:      for e ← m to |s'| do
5:        AP[] ← GeneratePossibleAlignments(b, m, e)
6:        for all A in AP[] do
7:          Add C = (s', b, m, e, A) to L
8:        end for
9:      end for
10:   end for
11: end for
```

Algorithm 1 outlines the technique used to assemble the primary library of contramers. As input serves a VNTR sequence s' over the alphabet of tandem repeats Σ'. The resulting list L contains each contramer possible under the described model. The *cost* of a contramer $C = (s', b, m, e, A)$ may be derived directly from the associated alignment A by adding the costs of mutations and excisions to the obligatory duplication cost:

$$\begin{aligned} cost(C) \;=\; & cost(dup(s'[b, m-1])) \\ & + \sum_{\substack{i=1,\ldots,|A| \\ A_{1i} \neq - \neq A_{2i}}} cost(mut(A_{1i}, A_{2i})) \\ & + \sum_{w \text{ excision in } A} cost(exc(w)). \end{aligned}$$

Theorem 2 (completeness of the primary library): Contramers contained in the primary library exhaustively generate all ancestor strings that can be derived from a VNTR sequence by reversing exactly one duplication event in an optimal transformation.

Proof: Algorithm 1 exhaustively generates all alignments for each pair of non-empty adjacent substrings of s', and hence each possibility of linking prefix repeats with suffix repeats including mutations and excisions are realized. Empty substrings are not considered as prefixes or suffixes of duplications since they cannot be part of an optimal transformation. ■

B. The secondary library

In order to infer cascaded duplication histories, overlapping contramers of the primary library are to be merged.

Two contramers $C_1 = (s', b_1, m_1, e_1, A_1)$ and $C_2 = (s', b_2, m_2, e_2, A_2)$ are *overlapping* if their *index sets* $ind(C_1) = \{b_1, \ldots, e_1\}$ and $ind(C_2) = \{b_2, \ldots, e_2\}$ share at least one common position (one tandem repeat) of s'.

The union of overlapping contramers forms a *merged contramer* representing a cascaded duplication event. Abusing notation, we denote by $lnk(C_1 \cup C_2)$ the result of merging two contramers whose *link sets* are $lnk(C_1)$ and $lnk(C_2)$, consisting of the aligned positions in A_1 and A_2, respectively.

If the intersection $ind(C_1) \cap ind(C_2)$ of two overlapping contramers comprises positions of both segments of C_1, we call C_1 a *contained* contramer and C_2 a *containing* contramer. All other pairs of overlapping contramers are *connected*. Note that not all overlapping duplication events are necessarily compatible with each other. The precondition for a pair of *compatible* contramers (C_1, C_2) is that they can be realized in a common *evolutionary order*, i.e., there exists at least one repeat history tree comprising both described duplication events. To describe the order of duplications, we use arrow symbols to relate the corresponding contramers. $C_1 \rightarrow C_2$ means that the duplication event captured by C_1 happened before C_2, whereas $C_1 \leftrightarrow C_2$ allows an arbitrary order of the duplications.

Observation 2 (evolutionary order): The common evolutionary realizability of the duplications represented by two contramers C_1 and C_2 can be deduced from analyzing the intersection $ind(C_1) \cap ind(C_2)$. If one contramer is contained in the other, the duplication events described by the contained contramer C_1, say, must have happened before the duplication events of the containing contramer C_2 ($C_1 \rightarrow C_2$, Fig. 3a). For a pair of connected contramers, the evolutionary order does not matter ($C_1 \leftrightarrow C_2$, Fig. 3b). Two contramers mutually contained in each other are not realizable in a common repeat

history (Fig. 3c), even if they share the same meridian position m (Fig. 3d).

Observation 2 results from the *transitivity* created when merging contramers: any common position $j \in ind(C_1) \cap ind(C_2)$ linked by A_1 to a position $i \notin ind(C_2)$ and by A_2 to a position $k \notin ind(C_1)$ transitively links $s'[i]$ to $s'[k]$. Obviously, for $i = k$ both contramers describe the same duplication of position j and therefore exclude each other from being included in a common history (i.e., mutually contained contramers in Observation 2). More generally, there has to exist a valid evolutionary order on the duplication events described by C_1 and C_2, that are $C_1 \rightarrow C_2$, $C_2 \rightarrow C_1$, or $C_1 \leftrightarrow C_2$. From this, we can straightforwardly formulate the criteria for merging two contramers.

Lemma 1 (pairwise merging conditions): Two contramers $C_1 = (s', b_1, m_1, e_1, A_1)$ and $C_2 = (s', b_2, m_2, e_2, A_2)$ are compatible and can be merged if:

1) they overlap, i.e. $ind(C_1) \cap ind(C_2) \neq \emptyset$, and
2) one of the reflected duplication events has happened after the other one. Therefore at least one contramer needs to have a segment outside of the intersection area, i.e., at least one of the positions $m_1 - 1, m_1, m_2 - 1, m_2$ must not be contained in $ind(C_1) \cap ind(C_2)$.

Proof: Let C_1 be a contramer that overlaps with C_2 such that $m_1 - 1, m_1 \in ind(C_1) \cap ind(C_2)$, then $C_1 \rightarrow C_2$ holds. Similarly, $m_2 - 1, m_2 \in ind(C_1) \cap ind(C_2)$ imposes the evolutionary order $C_2 \rightarrow C_1$ that clearly contradicts $C_1 \rightarrow C_2$. In the same manner, chains of connected contramers that impose contradicting evolutionary orders cannot be merged (Fig. 4). ■

Lemma 1 describes the preconditions that are to be met to merge a pair of contramers. If they intersect and they are compatible, C_1 and C_2 are merged into a new contramer by combining their respective alignments: the three repeats of every transitive link $(s'[i], s'[j], s'[k])$ can be written in a common column of the merged alignment (Fig. 5a). Problems arise when both contramers comprise excisions in between corresponding positions of the overlapping area (Fig. 5b). In this case C_1 and C_2 do not provide unique information about the transitive relation between the excised characters. One possibility would be to exhaustively generate all the alignment possibilities between the respective characters. However, since we are only interested in finding a "good" combination of characters minimizing the distance to another sequence, we leave these ambiguous repeats unaligned for the moment and search for the least-cost combination later on in the comparison step (Section IV).

The merging strategy is extendable to deal with more than two contramers. A set of combinable contramers $\{C_1, C_2, \ldots, C_r\}$ obviously requires that each of the contramers C_x must be connected to the other contramers in the set. Otherwise C_x is isolated and cannot be merged. Furthermore, it is required that each pair of overlapping contramers (C_x, C_y) is compatible. The next lemma demonstrates that the order in which the contramers are merged does not matter.

Lemma 2 (commutativity): The pairwise merging steps of multiply merged contramers are *commutative*.

Proof: Consider merging the alignments A_1 and A_2 of two contramers C_1 and C_2. Here, the same link $(s'[i], s'[k])$ in $lnk(C_1 \cup C_2)$ is created if a link $(s'[i], s'[j])$ from $lnk(C_1)$ is extended by a link $(s'[j], s'[k])$ from $lnk(C_2)$ or if the link $(s'[j], s'[k])$ from $lnk(C_2)$ is extended by the link $(s'[i], s'[j])$ in $lnk(C_1)$. ■

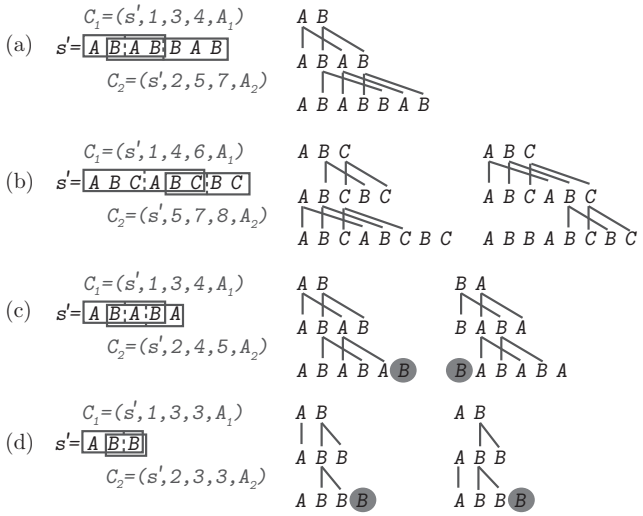


Fig. 3. Restrictions on compatible contramer pairs $C_1 = (s', b_1, m_1, e_1, A_1)$ and $C_2 = (s', b_2, m_2, e_2, A_2)$ (grey rectangles, meridian position indicated by a dashed line). The implied evolutionary order and possible repeat histories expressed by merging C_1 and C_2 are shown on the right with contradictions shaded in grey. (a) C_1 is *contained* in C_2, therefore the evolutionary order is fixed and the duplication captured in C_1 must have happened before the one described by C_2 (only one possible repeat history $C_1 \rightarrow C_2$). (b) Merging two *connected* contramers imposes no order on the evolution ($C_1 \leftrightarrow C_2$, the duplication of C_1 or C_2 may have happened first). (c) and (d) If none of the contramers has a non-intersecting segment, i.e. $m_1 - 1, m_1, m_2 - 1, m_2 \in ind(C_1) \cap ind(C_2)$, then *no repeat history* can be found incorporating both duplication events captured by the contramers. This holds even if the meridians coincide, $m_1 = m_2$, see (d).

Thus, when merging multiple contramers $C_1, C_2, \ldots, C_r$, with all preconditions met, we can perform the merging in any order successively for each pair of overlapping contramers (C_x, C_y), $x \neq y$ (Figure 6).

Algorithm 2 describes the construction of contramers in the secondary library. Initially, L comprises the contramers already included in the primary library. The set of contramers with beginning b, meridian m, and end e can be accessed via the function GETC(L, b, m, e). For chains of already merged contramers the variables b, m and e always refer to the contramer at the end (i.e., with the largest position e; ties are broken by smaller start position b). The functions FINDCONNECTEDC() and FINDCONTAINEDC() extract compatible contramers in a given subarea (specified by the start and end point). Compatibility is checked according to Lemma 1.

Theorem 3 (completeness of the secondary library): The secondary library contains all ancestor strings u' that can be derived from a sequence s' under the VNTR model of evolution by reversing one or more duplication events in an optimal transformation.

Proof: By the iteration order of Algorithm 2 over ascending start positions b, all (possibly already merged) contramers are considered in left-to-right order. Since for already partially

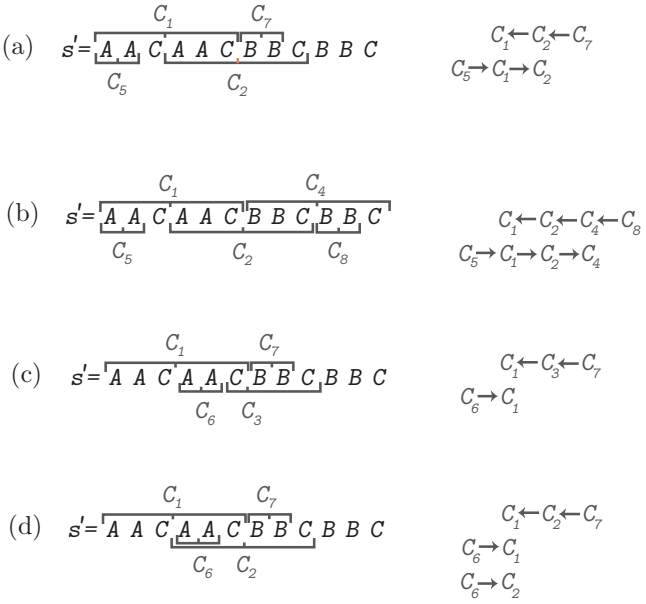


Fig. 4. Contained contramers and restrictions they put on the evolutionary order. Depicted are different subsets of the contramers $C_1 = (s', 1, 4, 6, A_1)$, $C_2 = (s', 4, 7, 9, A_2)$, $C_3 = (s', 6, 7, 9, A_3)$, $C_4 = (s', 7, 10, 12, A_4)$, $C_5 = (s', 1, 2, 2, A_5)$, $C_6 = (s', 4, 5, 5, A_6)$, $C_7 = (s', 7, 8, 8, A_7)$, $C_8 = (s', 10, 11, 11, A_8)$. Shown on the right are constraints imposed on the evolutionary order with contradictions shaded in grey. (a) C_1 containing C_5 is incompatible to contramer C_2 overlapping C_1 in the segment that is not intersecting with C_5 and containing itself an inner duplication C_7. (b) This restriction also holds for chains of connected contramers (C_2, C_4) with an inner duplication (C_8). (c) demonstrates that C_1 can be connected to C_3 (with the inner duplication C_7) in the same segment of the contained contramer C_6, as long as the connected contramer is not overlapping C_6 like C_2 (d).

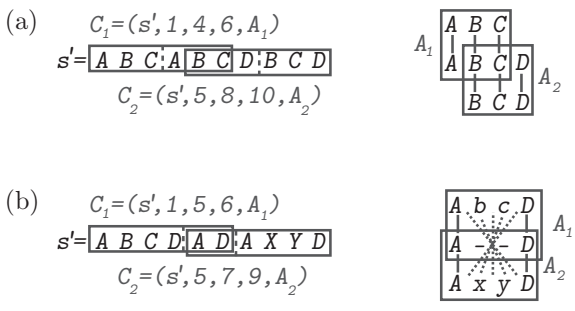


Fig. 5. Transitive links when merging contramers. (a) A pair of partially overlapping contramers where e.g. C_1 links positions 2 and 5, and C_2 links position 5 and 8. The transitive link created when merging A_1 with A_2 links all three B-characters together (2nd column of the merged alignment to the right). (b) Merging of contramers C_1 and C_2 that both induce characters in the same excised area. Consequently, the phylogenetic relation of the characters (lowercase) cannot be exactly determined (possible relations are indicated by the dotted grey lines).

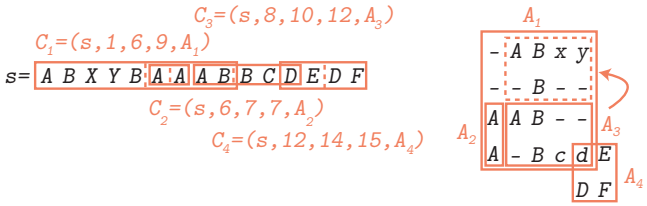


Fig. 6. A set of multiply merged contramers $\{C_1, C_2, C_3, C_4\}$ and the respective concatenated alignment. Note that lowercase characters are not uniquely aligned by the transitive links of the contramers, and their position is determined later during the comparison process (Section IV).

Algorithm 2 (Merge contramers to build the secondary library L)

```
1:  L ← PrimaryLibrary()
2:  for b ← 1 to |s'| − 1 do
3:    for e ← b + 1 to |s'| do
4:      for m ← b + 1 to e do
5:        CP[] ← GETC(L, b, m, e)
6:        for all C in CP[] do
7:          DP[] ← FINDCONTAINEDC(L, b, e; C)
8:          for all D in DP[] do
9:            F ← MERGE(C,D)
10:           ADD F TO CP[] AND L
11:         end for
12:         DP[] ← FINDCONNECTEDC(L, m, e; C)
13:         for all D in DP[] do
14:           F ← MERGE(C,D)
15:           ADD F TO L
16:         end for
17:       end for
18:     end for
19:   end for
20: end for
```

merged contramers $C_1, C_2, \ldots, C_r$ the iteration order refers to the contramer $C_x = (s', b_x, m_x, e_x, A_x)$ with largest end point e_x, always the contramer of a chain that was connected last is selected to be extended.

First, any compatible contramer $C_y = (s', b_y, m_y, e_y, A_y)$ *contained* anywhere in C_x is extracted. The product of merging C_x with C_y will not result in a chain of contramers that is iterated later on in the main loop. Therefore it has to be added at the end of the list $CP[]$ in order to allow multiple merges of C_x with different contained contramers.

Second, any compatible contramer $C_y = (s', b_y, m_y, e_y, A_y)$ *connected* to C_x is extracted. After merging the two, C_y will be iterated again later in the main loop since $b_y > m_x$. This results in a construction order of chains from the beginning of s' towards the end in which connected contramers that overlap with $s'[m_x, e_x]$ are contramers of the primary library that are not yet merged. Therefore, growing chains are extended by a single connected contramer in each cycle, in which afterwards possible inner duplications (contained contramers) are explored.

Thus, all valid combinations of contramers are found, and Lemma 2 respectively Theorem 2 complete the proof. ■

IV. VNTR Alignment

A. Contramer alignment

In the final alignment phase, the possible tandem repeat histories of two sequences s' and t' are used as alternative character combinations when comparing s' to t'. To this end, we extend the well established technique of dynamic programming (DP) for sequence alignment to additionally take into account the cascaded duplications. The contramers found along both sequences to be compared serve as additional alignment possibilities, i.e., cells extending the regular DP matrix. For each cell (i, j) to be computed in the DP recursion of the main matrix D with size $(|s'|+1)\times(|t'|+1)$, contramers ending at position i in s' (or at position j in t') are considered. The alignment A of each contramer $C = (s', b, m, e, A)$ can substitute the characters of the original sequence in the area $s'[b, e]$. For instance, the operation $align(s'[0, i], A)$ describes a submatrix of size $(i + 1) \times (|A| + 1)$ that substitutes the respective part of the original matrix of size $(i+1)\times(e-b)$. Therefore, the resulting alignment procedure needs to consider multiple alternative submatrices for each contramer in both sequences (Fig. 7). Matrix D is computed by the following recursion formula:

$$
\begin{aligned}
D(0,0) &= 0\\
D(0,j) &= cost(exc(t'[1,j])) \text{ for all } 1 \le j \le |t'|\\
D(i,0) &= cost(exc(s'[1,i])) \text{ for all } 1 \le i \le |s'|\\
D(i,j) &=
\end{aligned}
$$

$$
\min \begin{cases}
D(i-1,j-1) + cost(mut(s'[i],t'[j])) & \textit{//mutation}\\
D(x,j) + cost(exc(s'[x,i])) \text{ for all } x \le i & \textit{//excision in } s'\\
D(i,y) + cost(exc(t'[y,j])) \text{ for all } y \le j & \textit{//excision in } t'\\
D(x,b) + cost(C) + align(s'[x,i], A) & \\
\quad \text{for all } C = (t', b, m, j, A) \text{ and } x \le i & \textit{//duplication in } t'\\
D(b,y) + cost(C) + align(A, t'[y,j]) & \\
\quad \text{for all } C = (s', b, m, i, A) \text{ and } y \le j & \textit{//duplication in } s'\\
D(b_1,b_2) + cost(C_1) + cost(C_2) + align(A_1, A_2) & \\
\quad \text{for all } C_1 = (s', b_1, m_1, i, A_1) & \\
\quad \text{and } C_2 = (t', b_2, m_2, j, A_2) & \textit{//duplication in } s' \textit{ and } t'
\end{cases}
$$

Note that each cell (i, j) of the matrix D is connected by multiple contramers with any of the cells computed earlier during the DP process. We want to stress that when contramers substitute in both sequences, $align(A_1, A_2)$ spans a submatrix that is connected with D by 3 dependencies of its source cell (to $D(b_1 - 1, b_2 - 1), D(b_1, b_2 - 1)$ and to $D(b_1 - 1, b_2)$) and the connection to $D(i, j)$ in its sink cell (Fig. 7, left submatrix). However, $align(s'[x, i], A)$ (respectively $align(A, t'[y, j])$) span submatrices (Fig. 7, right submatrix) that are connected with $D(x, b)$ and $D(x - 1, b)$ (respectively $D(b, y)$ and $D(b, y - 1)$) in each row $x \le i$ (column $y \le j$). In an optimization, these submatrices are re-used and extended by each row (or column) with growing values of i (j).

Note that we do not need to explicitly consider the case that two excisions from adjacent contramers are fused because then also a larger (possibly merged) contramer exists that covers the complete excision.

At each stage (i, j) of the alignment, the minimum cost is calculated for all edit operations of the evolutionary model: mutation of repeats (comprising substitutions and indels on the DNA alphabet Σ), excisions of repeat copies (on the macro alphabet Σ') or duplication events (as described above). The order of computation of the algorithm should be performed as given. In our implementation, to optimize the performance we added a bounding step such that only alignments of contramers C ending in cell (i, j) are considered whose cost $cost(C)$ does not exceed the best cost found earlier for the cell $D(i, j)$.

Theorem 4 shows why the recursion finds the minimum distance of two VNTR clusters under the model of VNTR evolution.

Theorem 4 (correctness): The dynamic programming recurrence computes the alignment with minimal distance $d(s', t')$ between two VNTR sequences s' and t' under the VNTR model of evolution.

Proof: Initialization. The minimal cost between two empty strings is 0, between an empty string and a VNTR sequence it is the cost of excising this sequence.

Induction. We prove the correctness of the recurrence by decomposing the alignment of arbitrary prefixes $s'[0, i], 0 \le i \le |s'|$ and $t'[0, j], 0 \le j \le |t'|$ into the alignment of shorter prefixes $s'[0, x]$, $x < i$, and $t'[0, y], y < j$, and an additional operation. The latter is restricted to the operations of which the dynamic programming recurrence takes the minimum. Since within the recursion no further contractions are generated, it depends on the duplication events captured in the secondary library of contramers. The recursion therefore correctly retrieves $d(s', t')$, if and only if the contramers represent all valid evolutionary histories under the given model. This was shown in Theorem 3. ■

In order to analyze the time complexity of our method, we have to estimate the number of contramers generated in the earlier steps. Let $n = |s'|$ be the length of the given VNTR sequence. In the generation of the primary library, all $\mathcal{O}\left(n^3\right)$ combinations of beginning b, meridian m and end e such that $1 \le b < m \le e \le n$ are iterated. For each of these, all alignment possibilities are generated exhaustively. As shown in [21], for two sequences of length n there exist approximately $2^{2n}/\sqrt{\pi n}$ alignments, bounding the size of the primary library by $\mathcal{O}(n^3 2^{2n}/\sqrt{\pi n}) = \mathcal{O}\left(n^{2.5} 2^{2n}\right)$.

Let x be the size of the primary library. In the secondary library, new contramers are generated merging the contramers of the primary library. In the worst case, every subset of the x elements of the primary library forms a merged contramer in the secondary library. Hence an upper bound for the size of the second library is 2^x.

In the final step, the VNTR alignment, an alignment matrix has to be generated for each of the contramers of the second library as well as for the repeat sequences. Assuming now $n = \max\{|s'|, |t'|\}$, the main DP matrix as well as each of the submatrices have $\mathcal{O}(n^2)$ entries, each of which has $\mathcal{O}(n)$ predecessors. The averall computation time, however, also depends on the time required to calculate the cost of merged contramers during the alignment process, the subject of the next subsection.

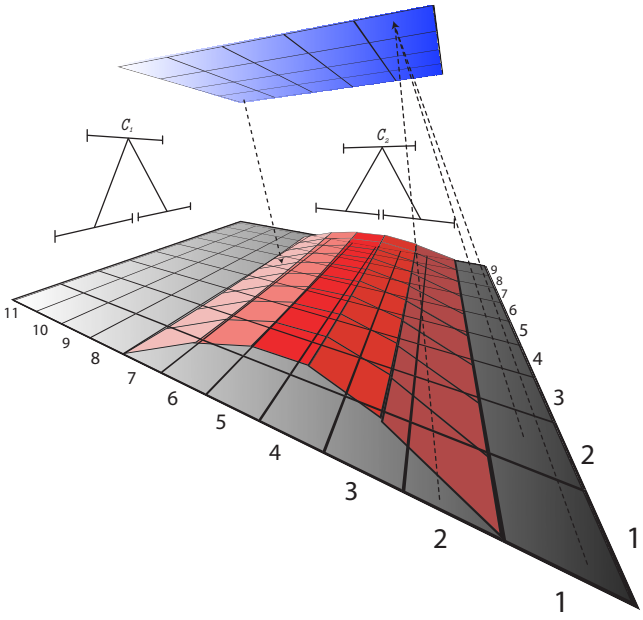


Fig. 7. Examples for alternative submatrices within a DP matrix D. The floating submatrix aligns $C_1 = (s', 2, 5, 6, A_1)$ with $C_2 = (t', 2, 5, 7, A_2)$. Dependencies of cell initialization with the main matrix D are depicted as dotted lines. The source cell of the submatrix is initialized by the $D(1,1)$, $D(1,2)$ and $D(2,1)$ while the result is used when determining the optimal cost in cell $D(6,7)$. In contrast, the bridge-like submatrix aligning $C_2 = (t', 2, 5, 7, A_2)$ to s' is connected to the main matrix at the beginning and the end of each of its rows (not shown). To be specific, all pathes from cells in column 1 of D are taken into account when determining optimal costs for the cells in the first column of the submatrix and on the other hand, pathes from all cells in the last column of the submatrix are respected for calculating the minimum costs in column 8 of D.

B. Costs of merged contramers

As observed earlier (Section III-B, Fig. 5b), in merged contramers not necessarily all of the transitive relations can uniquely be established. These positions are to be aligned within the merged contramer taking into account the sequence the contramer is compared to. Merged contramers may contain concatenated alignment areas of three or more rows, corresponding to transitive links in the overlapping area of contramers. The sum-of-pairs score normally used to measure multiple alignments is not suitable here since it does not reflect the order of the segments, i.e., the rows in the alignment; in a regular alignment, the sequences (the rows) are independent of their order while the rows of contramer-alignments represent an order of the evolutionary segments. The scoring function has to respect this order, and all possible repeat histories are to be explored to find the minimum cost for transitively linked areas.

Algorithm 3 formalizes the algorithm to find the minimum distance for merged contramers. To this end we use our implementation of the hyperspace multiple sequence alignment procedure [22], which was modified to use the scoring function for repeat evolution. The input is a set S of k sequences that correspond to the lines of two (possibly merged) contramer alignments A_1 and A_2, or to simple substrings of the VNTR sequences s' and/or t'. Let the number of lines from the first

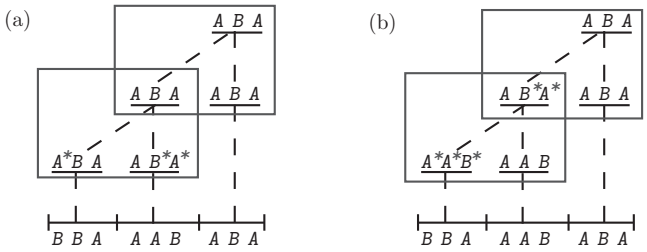


Fig. 8. Depicted is a repeat history for three repeats originating from two tandem duplication events (grey rectangles). Although the topology of the repeat history is the same in (a) and (b), the calculated costs diverge when changing the time point a certain mutation (grey asterisk) happens. In (a), three mutation events are used, whereas in (b), in total five mutations are performed. Note that the time a certain mutation takes place is independent for all positions of a duplication.

alignment or sequence be $x \geq 1$ and the number of lines from the second alignment or sequence be $k - x \geq 1$. For all pairs of positions (r_1, r_2) in the input, function ALIGNABLE(r_1, r_2) tests for alignability w.r.t. the constraints (i.e., the links) provided by the contramer alignments and stores alignable tuples (r_1, r_2) in a list F. Alignment of positions not contained in F is not considered later on.

Type **point** is a k-tuple of integer coordinates created by $p \leftarrow$POINT$(p_1, p_2, \ldots, p_k)$ such that $(q_1, \ldots, q_x) - (p_1, \ldots, p_x) \in \{0,1\}^x \setminus \{0^x\}$ and $(q_{x+1}, \ldots, q_k) - (p_{x+1}, \ldots, p_k) \in \{0,1\}^{k-x} \setminus \{0^{k-x}\}$. By this, all possible orders are iterated for repeat copies for which the transitive relation has not yet been fixed (Figure 5b). A **vertex** v stores, apart from its coordinate of the alignment matrix $v.P$, the sequence data $v.S$ on which the indices are based and the minimum cost $v.D$ found for a path from the source cell src to v. These costs are successively summed up for each column q. As explained later on, COSTCOLUMN() calculates the optimal mutation costs for repeats in the respective column of both of the compared contramers and MINCOST() returns the minimal cost to align one contramer column to the other. Later on, we will also introduce the conditions under which the evolutionary order of repeats in a column can be non-linear. In these instances PERMUTE() generates additional permutations of the input sequences.

The **priority queue** ranks the vertices by distance from src, where INSERT(Q, v) inserts a vertex at the appropriate position, DECREASE(Q,w) re-positions a vertex according to its new costs $w.D$, and EXTRACTMINIMUM(Q) returns the vertex v with the current minimum cost $v.D$. By this, the alignment is constructed node-by-node in a branch-and-bound fashion starting from the source vertex to the sink. By LOOKUP(v,Q) a vertex with the same coordinates and the same sequence data as v is returned, or $\perp$ if such an element is not contained in the queue. Under the assumption that the alignment is feasible, EMPTY(Q) will never become true and the algorithm terminates in line 13, returning the cost of an optimal alignment.

Figure 8 shows different examples for the evolutionary order in which mutation events could have taken place in merged contramers. As can be seen, this order is independent for neighboring repeats and therefore Algorithm 3 has to

Algorithm 3 (Exact alignment and cost calculation of characters in merged contramers.)

```
1: sequences S; int x; trie T; priority queue Q; vertex v, w; point p, q; list F
2: src ← VERTEX(0, POINT(0, 0, ..., 0), S)
3: snk ←VERTEX(∞,POINT(N_1, N_2, ..., N_K), ∅)
4: Q ← PRIORITY(); INSERT(Q, src)
5: for all (r_1, r_2) ∈ S do
6:    if ALIGNABLE(r_1, r_2) then
7:       F ← F + (r_1, r_2)
8:    end if
9: end for
10: while not EMPTY(Q) do
11:    v ← EXTRACTMINIMUM(Q); p ← v.P
12:    if v.P = snk.P then
13:       return v.D
14:    end if
15:    for all q ← POINT(q_1, q_2, ..., q_k)) such that (q_1, ..., q_x) −
       (p_1, ..., p_x) ∈ {0,1}^x \ {0^x} and (q_{x+1}, ..., q_k) −
       (p_{x+1}, ..., p_k) ∈ {0,1}^{k−x} \ {0^{k−x}}, (q_m, q_n) ∈ F and
       for all O ← PERMUTE(q) do
16:       w ← LOOKUP(q, Q)
17:       if w = ⊥ then
18:          w ← VERTEX(∞, q, O); INSERT(Q, w)
19:       end if
20:       if v.D+ COSTCOLUMN(p_{1,...,x}, q_{1,...,x}, O)+
          COSTCOLUMN(p_{x+1,...,k}, q_{x+1,...,k}, O)+
          MINCOST((q_1, ..., q_x)(q_{x+1}, ..., q_k)) < w.D then
21:          w.D ← v.D+ COSTCOLUMN(p_{1,...,x}, q_{1,...,x}, O)+
             COSTCOLUMN(p_{x+1,...,k}, q_{x+1,...,k}, O)+
             MINCOST((q_1, ..., q_x)(q_{x+1}, ..., q_k))
22:          DECREASE(Q, w)
23:       end if
24:    end for
25: end while
```

exhaustively investigate all possible combinations separately for each column of multiply merged contramers. Finding the best *order of mutation events* is equivalent to the problem of finding the best order of 1-duplications that produced a sequence of repeats. To solve it, we modify the so-called *arch technique* developed for minisatellite maps [13]. An arch is a set of ($\geq$ 2) positions on s' that all mutated from the same common ancestor repeat (*seed*). However, due to the more flexible cost scheme in the VNTR model, the optimality criterion from [13] has to be constrained as stated by the following lemma.

Lemma 3: An arch of length l is *optimal*, if (a) both marginal repeat types contain the seed A and if (b) it contains $a \geq \frac{l}{2}$ copies of A.

Proof: (a) Assume an arch with a marginal repeat $X_i \neq A$ and assume X_i is flanked by a repeat A inside the arch. Shortening the arch by one position removes $cost(mut(A, X_i)) \geq 0$ (b) Let an arch with a copies of A contain more tandem repeats $X_i, 1 \leq i \leq l-a$, with all $X_i \neq A$ and $X_{i1} \neq X_{i2}$. Assuming A as the seed of the arch, the total mutation costs are $\sum_{i=1}^{l-a} cost(mut(A, X_i))$, whereas any seed X_i generates $\sum_{j=1,j\neq i}^{l-a}(cost(mut(X_i, X_j))) + a\ cost(mut(X_i, A))$. By the triangle inequality assumption of mutation costs ($cost(mut(X_i, X_j)) + cost(mut(X_i, A)) \leq cost(mut(A, X_j))$), the relation $\sum_{i=1}^{l-a} cost(mut(A, X_i)) \leq \sum_{j=1,j\neq i}^{l-a}(cost(mut(X_i, X_j))) + a \cdot cost(mut(X_i, A))$ holds only if $a \geq \frac{l}{2}$. Therefore any arch with $a < \frac{l}{2}$ copies of its seed is not necessarily optimal. ■

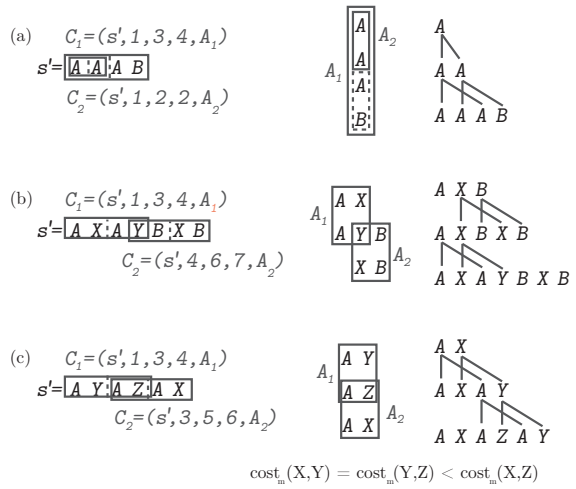


Fig. 10. Tandem duplication histories for contramer sets merged in different manners. Shown are the contramers on s' (left), the concatenated alignments (center), the constraints on the evolutionary order and an example of a repeat history (right). (a) A contained contramer C_1 always represents a duplication event that happened before the duplication of the contramer C_2 that it is contained in ($C_1 \rightarrow C_2$). Note that the history displayed is the only one possible for the merged contramers C_1 and C_2. (b) In partially overlapping contramers C_1 and C_2 the cost-optimal list of evolutionary steps is to be found ($C_1 \leftrightarrow C_2$). Every segment has to be investigated as possible root from where the repeats evolved in order of their location within the sequence. (c) When overlapping completely, finding the history of connected contramers is a tandem repeat history problem first described by Fitch [23]. In the history given here the last evolutionary segment (AX) evolves directly from the first one (AY). Note that for only partially overlapping contramers such a non-linear evolutionary order is not possible.

Figure 9a and b depicts how the best order of mutations is found independently for each column of merged alignments. Lemma 3 forces us to explore all possible mutation orders in between optimal arches in order to find minimum costs for a column (Figure 9b). This is performed in Algorithm 3 by

COSTCOLUMN$(p_{1,...,x}, q_{1,...,x}, O)+$
COSTCOLUMN$(p_{x+1,...,k}, q_{x+1,...,k}, O)+$
MINCOST$((q_1, ..., q_x)(q_{x+1}, ..., q_k))$.

COSTCOLUMN() yields the best order of mutations separately for the contramer columns of both sequences and MINCOST() determines the minimun cost to transform an arbitrary repeat copy from one sequence into one from the other sequence.

The arch technique modified according to Lemma 3 finds optimal mutation orders under the assumption that neighboring positions in a column are evolving consecutively from each other. While this condition always holds for *contained* contramers (Figure 10a), the evolutionary order of duplications in *connected* contramers can be *non-linear* (Figure 10c). Either, two merged contramers $C_1 = (s', b_1, m_1, e_1, A_1)$ and $C_2 = (s', b_2, m_2, e_2, A_2)$ *overlap partially*, s.t. $(m_1, e_1) \neq (b_2, m_2 - 1)$ and $(m_2, e_2) \neq (b_1, m_1 - 1)$. Here, some of the characters of C_1 or C_2 are not transitively linked with the other contramer, implying that the evolutionary order must be a subsequent chain, starting at any of the involved evolutionary fragments (Fig. 10b). Alternatively, two connected contramers may *overlap fully*, e.g. $(m_1, e_1) = (b_2, m_2 - 1)$. In this case

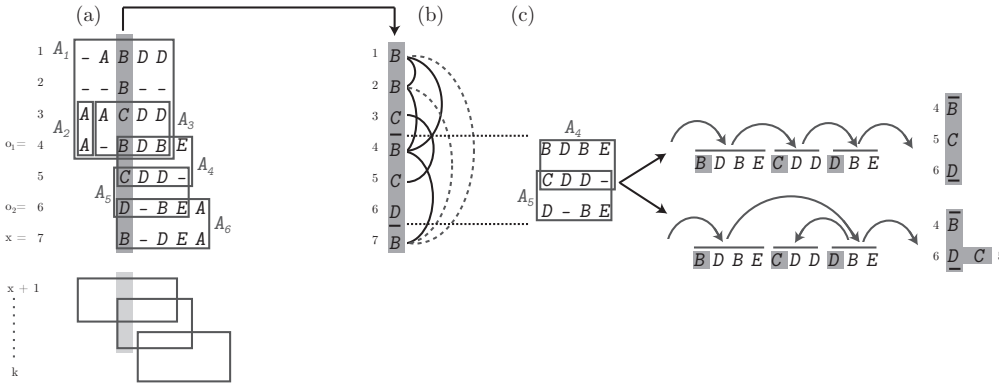


Fig. 9. Alignment of two multiply merged contramers, one comprising the lines 1 to $x = 7$ and the other one spanning lines $x + 1$ to k. The merged link set for the first sequence is given by $lnk(C_1 \cup C_2 \cup C_3 \cup C_4 \cup C_5 \cup C_6)$. (a) shows one column (grey) for which the minimum costs are to be determined. In (b) optimal (solid) and suboptimal arches (dashed) for one of the multiply merged contramers are depicted for the respective column. Note that the best order of mutations is found for each column independently in Algorithm 3). (c) Two possible orders for the duplications captured by the completely overlapping contramers C_4 and C_5 ($o_1 = 4, o_2 = 6$). In contrast to the order of mutations, the order of duplications within completely overlapping contramers stays the same for all included positions (columns of the alignment). These orders are generated by PERMUTE($\{o_1, \ldots, o_2\}$) in Algorithm 3. Corresponding parts from the alignment column shown in (b) are given to the right of the corresponding duplication order.

the evolutionary order is arbitrary and all possible duplication tree topologies have to be investigated to find the history with least cost (Fig. 10c). To yield minimum costs, we therefore separate the problem of finding the best *order of duplications* in completely overlapping contramers from the problem of finding the best *order of mutation events*. Algorithm 3 implements this as follows: if the first position of fully overlapping contramers is included in a newly constructed point q, possible duplication orders O are correspondingly added as alternatives by PERMUTE(q). Unlike the mutation order, the order of duplications obviously stays the same for all positions involved in the same fully overlapping duplication events.

V. RESULTS

To test our method, we applied it to the DNA sequences of *Staphylococcus aureus*. To be specific, the 5'-VNTR clusters in the gene encoding the *spa*-Protein were used as input for pairwise alignment under the model of evolution described in Section II. Since the *spa* types for all hitherto isolated strains are known, the sequences are provided in characters of the macro alphabet Σ'. To this point, we use the *Kreiswirth* notation defined by $\Sigma' = \{-, A, A_2, B, B_2, C, C_2, \ldots, V, V_2, W, X, Y, Y_2, Z, Z_2\}$. In addition to the simplified alphabet used to introduce the model, in the Kreiswirth notation each letter may be used more than once in conjunction with a unique index [4].

Figure 11 shows an alignment of the *spa* repeat patterns that we used to set up a simple cost scheme for the comparison of *spa* types: since we are interested in a distance to measure evolutionary steps, we assign a unit cost u. In our tests, we always set $u = 1$. The objective function to score *mutations* is based on a multiple alignment of all repeat types (Fig. 11). In order to contribute to the fact that the repeat cluster is coding, nucleotide substitutions changing effectively the corresponding codon are weighted with a cost of u, while silent mutations are omitted. In the same manner indels are penalized according to the number of codons x missing (xu). The mutation costs $cost(mut(a, b))$ for $a, b \in \Sigma' \setminus \{-\}$ are summed up along the pairwise DNA alignment of a and b which is projected from the global alignment of all repeats. A *duplication* has always unit cost, regardless of its degree. An *excision* is treated differently, we penalize it according to its length, such that $cost(exc(s'[i_1, i_2])) = (i_2 - i_1)$. The linear cost function prevents the algorithm from replacing all evolutionary events by excisions when repeat copies are no longer exact. From another point of view, the scoring biases the algorithm to favor duplications and mutations and prefer them – up to a certain threshold – over possible excisions.

Since, to our knowledge, this is the first time the VNTR data of *spa* types is used to infer distance measures, we focus on one sequence type (ST), ST-254, that by definition pools strains with the same types of the seven housekeeping genes used for MLST [25]. However, the resolution of STs found by MLST is lower than the microvariation within the *spa* repeat cluster. Thus, a ST group with an identical MLST pattern can pool several strains with diverging *spa* types (named by "t" and a 3-digit code), while a *spa* type may have evolved in different ST groups. *spa* types used in here to investigate the microvariation of the repeats (i.e., t036, t048, t115, t139, and t146) were isolated in the laboratory from identical strain stocks [24]. Therefore, the microvariation of these *spa* types can be assumed to bear a phylogenetic marker (Fig. 12a).

Figure 12b summarizes the differences of applying the VNTR distance, compared to a standard distance measure that takes into account only substitutions and indels. In order to compare the results, we adapted the scoring function to the same values given for the VNTR evolution (xu for the insertion of x gaps and substitution costs according to non-synonymous mutations, as shown in Figure 11). Note that the alignments shown in Figure 12 are only one example from a set of alignments that can reproduce the minimal costs shown. As can be seen clearly, the distribution of the costs varies substantially between the described model and a model exclusively based on substitutions and indels. This results from

spa Repeat Code	1	2	3	4	5	6	7	8
A	AAA	GAA	GAC	AAC	AAA	AAA	CCT	GGC
A_2	GAG	GAA	GAC	GGC	AAC	AAA	CCT	GGT
B	AAA	GAA	GAC	AAC	AAA	AAA	CCT	GGT
B_2	AAA	GAA	GAT	AAC	AAC	AAG	CCT	GGT
C	AAA	GAA	GAC	AAC	AAA	AAG	CCT	GGC
C_2	AAA	GAA	GAC	AAT	AAC	AAG	CCT	GGT
D	AAA	GAA	GAC	AAC	AAC	AAA	CCT	GGC
D_2	GAG	GAA	GAC	AAT	AAC	AAA	CCT	GGT
E	AAA	GAA	GAC	AAC	AAC	AAA	CCT	GGT
E_2	AAA	GAA	GAC	AAT	AAC	AAG	CCT	GGT
F	AAA	GAA	GAC	AAC	AAC	AAG	CCT	GGC
F_2	AAA	GAA	GAC	AAC	AAA	AAG	CCT	AGC
G	AAA	GAA	GAC	AAC	AAC	AAG	CCT	GGT
G_2	AAA	GAA	GAC	AGC	AAC	AAG	CCT	GGC
H	AAA	GAA	GAC	AAT	AAC	AAG	CCT	GGC
H_2	AAA	GAA	GAT	GGC	AAC	AAG	CGT	AGT
I	AAA	GAA	GGC	AAC	AAA	AAA	CCT	GGT
I_2	GAG	GAA	GAC	AAC	AAC	AAA	CCT	GGC
J	AAA	GAA	GAC	GGC	AAC	AAA	CCT	GGC
K	AAA	GAA	GAC	GGC	AAC	AAA	CCT	GGT
L	AAA	GAA	GAC	GGC	AAC	AAG	CCT	GGC
M	AAA	GAA	GAC	GGC	AAC	AAG	CGT	GGT
N	AAA	GAA	GAT	GGC	AAC	AAA	CGT	GGC
O	AAA	GAA	GAT	GGC	AAC	AAA	CCT	GGT
P	AAA	GAA	GAT	GGC	AAC	AAG	CGT	GGC
Q	AAA	GAA	GAT	GGC	AAC	AAG	CCT	GGT
R	AAA	GAA	GAT	GGT	AAC	AAA	CCT	GGC
T	GAG	GAA	GAC	AAC	AAA	AAA	CCT	GGT
U	GAG	GAA	GAC	AAC	AAC	AAA	CCT	GGT
U_2	CAA	GAA	GAC	GGC	AAC	AAG	CCT	GGT
V	GAG	GAA	GAC	AAC	AAC	AAG	CCT	AGC
V_2	CAA	GAA	GAC	AAC	AAC	AAG	CCT	GGT
W	GAG	GAA	GAC	AAC	AAC	AAG	CCT	GGC
X	GAG	GAA	GAC	AAC	AAG	AAG	CGT	GGT
Y	GAG	GAA	GAC	AAT	AAC	AAG	CCT	GGC
Y_2	GAG	GAA	GAC	AAC	AAA		CCT	GGC
Z	GAG	GAA	GAC	AAT	AAC	AAG	CCT	GGT
Z_2	AAA	GAA	GAC	AAC		AAG	CCT	GGT

AAA, AAG (Lys) GAC, GAT (Asp) GGC, GGT (Gly) CCT (Pro)
GAA, GAG (Glu) AAC, AAT (Asn) AGC, AGT (Ser) CAA (Gln)

Fig. 11. DNA sequences of the *spa* repeat types known hitherto. To the left, the letter denoting the repeat type in the *Kreiswirth* notation is given. The 24-bp repeats are grouped in triplets according to the reading frame. Shaded boxes indicate the different amino acids that are translated from the codons. Data adapted from [4].

(a) *ST254:*

t036 YGFMBQBLPO
t048 YGFMBLPO
t115 YGFMBQBLQBLQBLPO
t139 YFMBQBLQBLPO
t146 YFMBQBPO

(b)

t036 YGFMBQBLPO
t048 YGFM--BLPO
cost=1 (2)

t048 YGFMBL-----PO
t139 Y-FMBQBLQBLPO
cost=3 (6)

t036 YGFMBQBL------PO
t115 YGFMBQBLQBLQBLPO
cost=2 (6)

t048 YGFMBL-PO
t146 Y-FMBQBPO
cost=2 (2)

t036 YGFMBQBL---PO
t139 Y-FMBQBLQBLPO
cost=2 (4)

t115 YGFMBQBLQBLQBLPO
t139 Y-FMBQBLQBL---PO
cost=2 (4)

t036 YGFMBQBLPO
t146 Y-FMBQB-PO
cost=2 (2)

t115 YGFMBQBLQBLQBLPO
t146 Y-FMBQB-------PO
cost=4 (8)

t048 YGFMBL--------PO
t115 YGFMBQBLQBLQBLPO
cost=3 (8)

t139 YFMBQBLQBLPO
t146 YFMBQB----PO
cost=2 (4)

Fig. 12. Sequence comparison of the MLST sequence type ST-254. (a) a list of *spa* types found to have the ST-254 pattern. The data was acquired by sequencing from the same laboratory strains the VNTR cluster of the Spa protein and the MLST loci [24]. (b) One alignment that scores minimal costs for each pair of *spa* types from the ST-254 group. Substrings involved in duplication events leading to the minimum distance are underlined. To the right of the alignments the costs are given w.r.t. the *spa* type model and in parentheses the costs under a model that is not taking into account duplications. Under the evolutionary model, the costs in the comparison of $t036$ and $t048$ are composed of a duplication event of the substring $t048[5,6] = BL$ and the mutation of repeat L into $t036[6] = Q$, yielding the distance $d(t036, t048) = cost(dup(t048[5,6]))+ cost(mut(L,Q))= 1+0=1$.

the simple fact that the latter scoring scheme is not capable to distinguish duplication events from excisions and cannot score them differently. The resolution of the VNTR model is therefore higher, which results in a more detailed analysis when comparing the distances.

For instance, the minimal costs of the alignments in Fig. 12b can be calculated as follows. (Note that mutation costs of cost 0 are omitted. Therefore, character changes are possible between repeats that do only differ by silent mutations.)

$$
\begin{aligned}
d(t036,t048) &= cost(dup(t048[5,6])) + cost(mut(L,Q)) = 1\\
d(t036,t115) &= 2cost(dup(t036[6,8])) = 2\\
d(t036,t139) &= cost(dup(t036[6,8])) + cost(exc(t036[2,2])) = 2\\
d(t036,t146) &= cost(dup(t036[2,2])) + cost(exc(t036[8,8])) = 2\\
d(t048,t115) &= cost(dup(t048[5,6])) + 2cost(dup(QBL)) = 3\\
d(t048,t139) &= cost(dup(t048[2,2])) + cost(dup(t048[4,5]))\\
&\quad +cost(dup(QBL)) = 3\\
d(t048,t146) &= cost(dup(t048[2,2])) + cost(exc(t146[6,6])) = 2\\
d(t115,t139) &= cost(dup(t115[2,2])) + cost(exc(t115[6,8])) = 2\\
d(t115,t146) &= cost(dup(t146[2,2])) + cost(exc(t115[8,8]))\\
&\quad +2cost(dup(QBL)) = 4\\
d(t139,t146) &= cost(dup(t139[7,7])) + cost(dup(QBL)) = 2
\end{aligned}
$$

VI. Conclusion

The VNTR model of evolution joins point mutations (substitutions and indels) and mutations that involve whole tandem repeats (excisions and duplications of variable degree) while allowing them in arbitrary order. To our knowledge, this is the first time an evolutionary model containing duplications of variable degree has been described for sequence comparison. Note that, because *spa* repeats are within a coding region, we restricted our VNTR model to single-step duplications and excisions at the boundaries of the repeat types of Σ'. Taking into account operations as captured in the VNTR model, we described an exact method to compare a pair of repeat sequences and to assign them a distance. In first tests we could show that the pairwise comparison under the VNTR evolution efficiently captures cascades of duplication events and expresses them in the distance measure. Traditional scoring functions based on a model restricted to substitutions and indels cannot resolve duplication events, and scoring schemes that additionally consider duplications (but no excisions) do not support cascaded cycles of duplications, excisions and mutations. However, *in vivo* studies demonstrated how essential these mechanisms are, when investigating the evolution of *S. aureus* [5].

Acknowledgment

The authors would like to thank France Denoeud and Dag Harmsen for fruitful discussions.

References

[1] P. L. Flèche, Y. Hauck, L. Onteniente, A. Prieur, F. Denoeud, V. Ramisse, P. Sylvestre, F. Ramisse, and G. Vergnaud, "A tandem repeats database for bacterial genomes: application to the genotyping of *Yersinia pestis* and *Bacillus anthracis*," *BMC Microbiol.*, vol. 1, p. 2, 2001.

[2] R. Fronthingham and W. Meeker-O'Connell, "Genetic diversity in the mycobacterium tuberculosis complex based on variable numbers of tandem dna repeats," *Microbiology*, vol. 144, pp. 1189–1196, 1998.
[3] P. L. Flèche, M. Fabre, F. Denoeud, J.-L. Koeck, and G. Vergnaud, "High resolution, on-line identification of strains from the *Mycobacterium tuberculosis* complex based on tandem repeat typing," *BMC Microbiol.*, vol. 2, p. 37, 2002.
[4] L. Koreen, S. V. Ramaswamy, E. A. Graviss, S. Naidich, J. M. Musser, and B. N. Kreiswirth, "spa typing method for discriminating among staphylococcus aureus isolates: implications for use of a single marker to detect genetic micro- and macrovariation," *J. Clin. Microbiol.*, vol. 47, pp. 792–799, 2004.
[5] B. Kahl, A. Mellmann, S. Deiwick, G. Peters, and D. Harmsen, "Variation of the polymorphic region X of the protein A gene during persistent airway infection of cystic fibrosis patients reflects two independent mechanisms of genetic change in *Staphylococcus aureus*," *J. Clin. Microbiol.*, vol. 43, pp. 502–505, 2005.
[6] N. N. I. S. System, "National nocosomial infections surveillance (nnis) system report, data summary from january 1990-may 1999," *Am. J. Infect. Control*, vol. 27, pp. 520–532, 1999.
[7] M. de Macedo Brgido, C. R. M. Barardi, C. A. Bonjardin, C. L. S. Santos, M. de Lourdes Junqueira, and R. R. Brentani, "Nucleotide sequence of a variant protein a of *Staphylococcus aureus* suggests molecular heterogeneity among strains," *J. Basic Microbiol.*, vol. 31, pp. 337–345, 1991.
[8] A. van Belkum, S. Scherer, L. van Alphen, and H. Verbrugh, "Short-sequence dna repeats in prokaryotic genomes," *Microbiol. Mol. Biol. Rev.*, vol. 62, pp. 275–293, 1998.
[9] D. A. Robinson and M. C. Enright, "Evolutionary models of the emerge of methicillin-resistant staphylococcus aureus," *Antimicrob. Agents Chemother.*, vol. 47, pp. 3926–3934, 2003.
[10] R. Groult, M. Lonard, and L. Mouchard, "A linear algorithm for the detection of evolutive tandem repeats," in *The Prague Stringology Conference 2003*, 2003.
[11] D. Gusfield and J. Stoye, "Linear time algorithms for finding and representing all the tandem repeats in a string," *J. Comput. Syst. Sci.*, vol. 69, no. 4, pp. 525–546, 2004.
[12] G. Benson, "Sequence alignment with tandem duplication," *J. Comput. Biol.*, vol. 4, pp. 351–367, 1997.
[13] S. Bèrard and E. Rivals, "Comparison of minisatellites," *J Comput Biol*, vol. 10, pp. 357–372, 2003.
[14] B. Behzadi and J.-M. Steyaert, "The minisatellite transformational problem revisted," in *Proc. of WABI 2004*, ser. LNBI, vol. 3240, 2004, pp. 310–320.
[15] G. Benson and L. Dong, "Reconstructing the duplication history of a tandem repeat," in *Proc. of ISMB 1999*, 1999, pp. 44–53.
[16] D. Jaitly, P. Kearney, G.-H. Lin, and B. Ma, "Reconstructing the duplication history of tandemly repeated genes," *J. Comput. Sys. Sci.*, vol. 65, pp. 494–507, 2002.
[17] O. Elemento, O. Gascuel, and M.-P. Lefranc, "Reconstructing the duplication history of tandemly repeated genes," *Mol. Biol. Evol.*, vol. 19, pp. 278–288, 2002.
[18] D. Bertrand and O. Gascuel, "Topological rearrangements and local search method for tandem duplication trees," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 2, pp. 1–13, 2005.
[19] M. Sammeth, T. Weniger, D. Harmsen, and J. Stoye, "Alignment of tandem repeats with excision, duplication, substitution and indels (EDSI)," in *Proceedings of the 5th International Workshop on Algorithms in Bioinformatics, WABI 2005*, ser. LNBI, vol. 3692, 2005, pp. 276–290.
[20] E. Rivals, "A survey on algorithmic aspects of tandem repeats evolution," *Int. J. Found. Comput. S.*, vol. 15, pp. 225–257, 2004.
[21] M. Waterman, *Introduction to computational biology: maps, sequences, and genomes*. CRC Press, 1995.
[22] M. Sammeth, J. Rothgänger, W. Esser, J. Albert, J. Stoye, and D. Harmsen, "Qalign: quality based multiple alignments with dynamic phylogenetic analysis," *Bioinformatics*, vol. 19, pp. 1592–1593, 2003.
[23] W. Fitch, "Phylogenies constrained by cross-over process as illustrated by human hemoglobins in a thirteen-cycle, eleven amino-acid repeat in human apolipoprotein a-i," *Genetics*, vol. 86, pp. 623–644, 1977.
[24] "Ridom nomenclature server," http://www.ridom.de/spaserver/nomenclature.shtml.
[25] M. Enright, N. Day, C. Davies, S. Peacock, and B. Spratt, "Multilocus sequence typing for characterization of methicillin-resistant and methicillin-susceptible clones of *Staphylococcus aureus*," *J. Clin. Microbiol.*, vol. 38, pp. 1008–1015, 2000.

Michael Sammeth studied Biology and Computer Science at the University of Würzburg. Supported by a fellowship of the Ernst Schering Research Foundation, he researched on special applications of multiple sequence alignment at Bielefeld University and the Center for Integrated Bioinformatics in Amsterdam. He received his PhD degree from the Technical Faculty of Bielefeld University in 2005 and he currently holds a postdoctoral position in Barcelona at the Center for Genome Regulation in Barcelona supported by the German Academic Exchange Service.

Jens Stoye studied applied computer science in the natural sciences at Bielefeld University, where he received the PhD degree in 1997 on a topic related to multiple sequence alignment. After postdoctorate positions at the University of California at Davis and the German Cancer Research Center in Heidelberg, he became head of the Algorithmic Bioinformatics group at the Max Planck Institute of Molecular Genetics in Berlin. Since 2002, Dr. Stoye has been a professor of genome informatics at Bielefeld University.

Abstract

The thesis presents enhancements for automated and manual multiple sequence alignment: existing alignment algorithms are made more easily accessible and new algorithms are designed for difficult cases.
Firstly, we introduce the QAlign framework, a graphical user interface for multiple sequence alignment. It comprises several state-of-the-art algorithms and supports their parameters by convenient dialogs. An alignment viewer with guided editing functionality can also highlight or print regions of the alignment. Also phylogenetic features are provided, e.g., distance-based tree reconstruction methods, corrections for multiple substitutions and a tree viewer. The modular concept and the platformindependent implementation guarantee an easy extensibility.
Further, we develop a constrained version of the divide-and-conquer alignment such that it can be restricted by anchors found earlier with local alignments. It can be shown that this method shares attributes of both, local and global aligners, in the quality of results as well as in the computation time. We further modify the local alignment step to work on bipartite (or even multipartite) sets for sequences where repeats overshadow valuable sequence information. In the end a technique is established that can accurately align sequences containing eventually repeated motifs.
Finally, another algorithm is presented that allows to compare tandem repeat sequences by aligning them with respect to their possible repeat histories. We describe an evolutionary model including tandem duplications and excisions, and give an exact algorithm to compare two sequences under this model.

Cover page

Kangaroo rat (*Dipodomys spec.*) escaping constrained dynamic programming matrices. The 30 cm (incl. tail) animal received its name from the ability to jump up to 2 meters when in danger. In the context of this thesis, it is noticeable that more than half of the kangaroo rat genome consists of families of just three basic repeats, with over two billion copies of a three-base repeat, two billion copies of a six-base repeat, and a billion copies of a ten-base repeat.